Cardinality Constrained Discrete-Time Linear-Quadratic Control

Gao Jianjun

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Systems Engineering and Engineering Management

July 2005

Acknowledgement

I would like to express my sincere thanks to my supervisor, Professor Duan Li, for his patient guidance of the development of the ideas in this research and in the writing of this thesis.

I am also thankful to Professor Erwei Bai for serving as the External Examiner of my thesis and to Professor XunYu Zhou and Professor Jie Huang for serving as members of my thesis committee.

I am grateful to Professor XiaoLin Sun for teaching me programming and debugging techniques and to my friends Jiang Xie and Yongwei Huang for many stimulating discussions about research ideas.

Finally, I would like to thank my parents and my girlfriend for their constant love and support. I have greatly enjoyed my study in the Department of Systems Engineering and Engineering Management of the Chinese University of Hong Kong. I would like to thank all of the professors, staff members, and my fellow students for their guidance, help, and friendship.

Abstract

Set-up cost is an indispensable factor that must be considered in many situations when implementing control actions. However, this factor has been neglected in the literature of control theory and methodology. This thesis considers a discrete-time linear-quadratic model with a limited control implementation time, which is known as the cardinality constrained discrete time linear-quadratic control problem. Efficiently solving this problem plays a key role in the solution of discrete-time linear-quadratic control problems with set-up cost. Dynamic programming is only practically useful in the scalar state case of the cardinality constrained linear quadratic control problem, and thus to handle the multi-dimensional state case, the problem is converted to a cardinality constrained quadratic programming problem. Efficient branch and bound rules are developed in this thesis to solve such a problem.

Keywords: linear-quadratic control, quadratic programming, cardinality constraint.

摘要

启动每次控制所付出的费用，我们称之为启动费用，在很多情况下是不可以被忽略的因素。然而这一因素在目前的控制理论及控制方法中并没有被涉及到。在这篇论文中，我们讨论了对控制次数有约束的离散线性二次型模型，我们称之为控制次数受限的最优线性二次型控制问题。如何有效地解决这一问题，是解决考虑有启动费用的离散线性二次型最优控制问题的关键。对于控制次数受限的最优线性二次型控制问题，传统的动态规划只能实际解决状态变量是一维的情况。对于状态变量是多维的情况，我们把原有的控制问题转变成一个决策变量个数受限的二次规划问题。本论文提出一些有效的分支定界方法来解决这个特殊的二次规划问题。

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Although investors have the freedom to adjust their portfolios at all times, they do not perform such adjustments in every period due to the transaction cost. Similarly, in many situations the set-up cost cannot be neglected. However, set-up cost is not taken into consideration in traditional control theory. Without a doubt, linear-quadratic control [1] [2] [3] [4] is one of the most successful developments of control theory, because of its wide application and mathematical elegance in tractability.

Consider the following discrete-time linear-quadratic control problem with a set-up cost that is attached to nonzero control actions.

$$\begin{aligned} (P) \qquad \min \quad & w\sum_{t=0}^{T-1} Supp(u_t) + x_T' Q_T x_T \\ & + \sum_{t=0}^{T-1} [x_t' Q_t x_t + u_t' R_t u_t], \\ \text{subject to :} \quad & x_{t+1} = A_t x_t + B_t u_t \\ & x_0 \text{ is given,} \end{aligned}$$

where $x_t \in R^n$ is the state, $u_t \in R^m$ is the control, $A_t \in R^{n\times n}$, $B_t \in R^{n\times m}$, for all t matrices $Q_t \in R^{n\times n}$ and $R_t \in R^{m\times m}$ are positive semi-definite and positive definite, respectively, $Supp(u_t) = 0$ if u_t is a zero vector and $Supp(u_t) = 1$ otherwise, and w is the set-up cost of implementing a control action.

When w is zero, problem (P) reduces to the conventional linear-quadratic optimal control problem, whereas a very large w forces all u_t, $t = 0, 1 \ldots, T-1$, to take a zero value. In general, the incorporation of a set-up cost into the formulation of linear-quadratic control is equivalent to placing an upper limit on the number of times that the control can be applied in discrete-time systems. Consider the following cardinality constrained discrete-time linear-quadratic control problem with a value of s that satisfies $1 \leq s \leq T$.

$$
\begin{aligned}
(P_s) \qquad & J(s) = \min x_T' Q_T x_T + \sum_{t=0}^{T-1} [x_t' Q_t x_t + u_t' R_t u_t] \\
& \text{Subject to : } x_{t+1} = A_t x_t + B_t u_t \\
& x_0 \text{ is given} \\
& \sum_{t=0}^{T-1} Supp(u_t) \leq s.
\end{aligned}
$$

When $s = T$, problem (P_s) reduces to the conventional linear-quadratic control problem. Let $v(\cdot)$ denote the minimum value of problem $(\cdot)$.

Lemma 1.1 *Monotonicity. For any k_1 and k_2 that satisfy $1 \leq k_1 < k_2 \leq T$, $v(P_{k_1}) \geq v(P_{k_2})$.*

Proof. The lemma is obvious from the fact that the feasible region S_{k_1} of $\{u_t\}\ |_{t=0}^{T-1}$ in problem (P_{k_1}) is a subset of the feasible region S_{k_2} of $\{u_t\}\ |_{t=0}^{T-1}$

in problem (P_{k_2}), that is,

$$S_{k_1} = \{\{u_t\} |_{t=0}^{T-1} \in R^{mT} \mid \sum_{t=0}^{T-1} Supp(u_t) \leq k_1\}$$

$$\subset \{\{u_t\} |_{t=0}^{T-1} \in R^{mT} \mid \sum_{t=0}^{T-1} Supp(u_t) \leq k_2\} = S_{k_2}.$$

In problem (P_s), the larger the cardinality size s, the smaller the optimal value $J(s)$. However, as s is increasing, the set-up cost also increases (see Figure 1.1). The optimal solution for the primary problem (P) can be found first by solving (P_s) for $1 \leq s \leq T$, and then identifying the optimal cardinality s^*,

$$s^* = arg \min_{0 \leq s \leq T} ws + J(s).$$

The solution to (P_{s^*}) is the optimal control of (P), and thus efficiently solving problem (P_s) plays a key role in solving problem (P).

There are no studies in the literature that directly discusses cardinality constrained optimal control problems. However, problem (P_s) has a certain relationship with the optimal control problem of switched systems where the system switches between two sub-control models, one of which has no control and one of which has control. The target of this optimal control problem is not only to find the optimal control input for the subsystem model, but also to find the optimal switching instance between these subsystems. Many real-world processes, such as chemical processes, automotive systems, and manufacturing processes, can be modeled using this modeling approach. The main approach to the solution of the optimal control problem of switching systems in the literature [8] [9] is to decompose the problem into two sub-problems by parameterizing the time interval of the switching instance, and then solving the lower lever problem using those

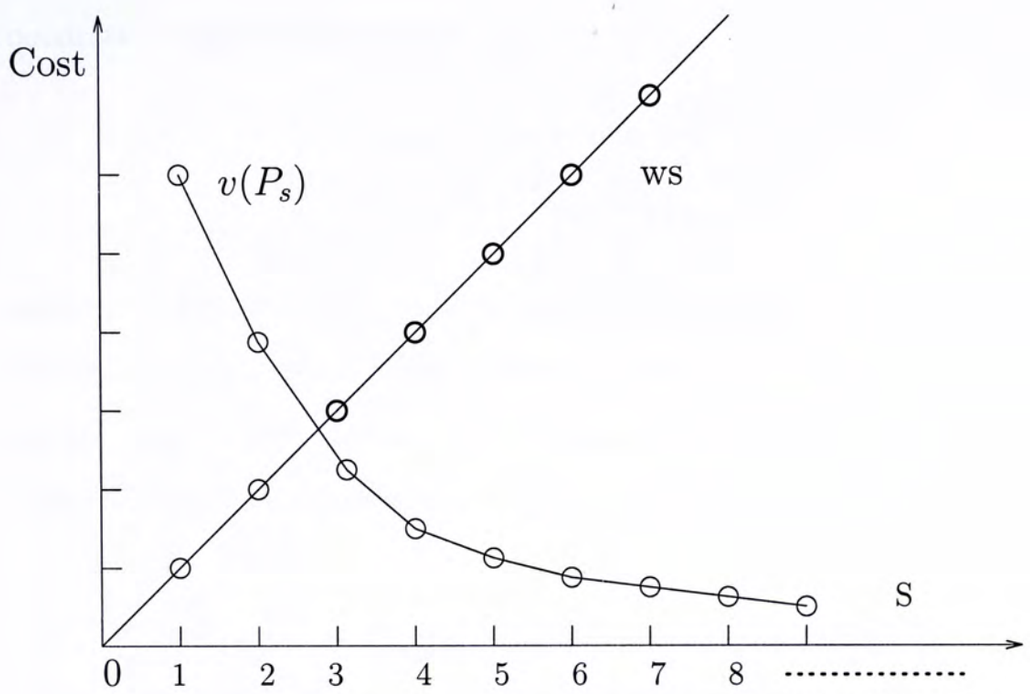


Figure 1.1: Cost with Different Values of s

parameters. At a higher lever, the switching instance is determined by nonlinear programming techniques. It must be pointed out that even for linear systems with a quadratic performance measure, an analytical solution for the lower level problem is not obtainable. Thus, the linear quadratic optimal control problem of switched systems remains as an open problem (see [7]), and the model in this thesis is just related to such a problem.

A natural connection between the linear-quadratic control problem and the quadratic programming problem can be easily revealed, if state x_t at any stage t is rewritten as a function of u_0,u_1,. . .,u_{t-1}. Using a matrix form, problem (P_s) can be expressed as a special case of the following cardinality constrained

quadratic programming problem (Q_s),

$$(Q_s) \qquad v(y) = \min \ \frac{1}{2} y'Gy + g'y$$
$$\text{Subject to : } \ Supp(y) \leq s,$$

where $y \in R^S$, G is an $S \times S$ positive definite matrix, g is an S-dimensional vector, $Supp(y)$ is the cardinality of y that is the number of components of y that are not equal to zero, and $s \leq S$ is a given positive integer number. The details of the transformation procedure are given in Section 3.1.

Problem (Q_s) is studied in [10], in which the mathematic model contains bounds that have been placed for components of y and there is a linear side constraint $Ay \leq b$. It is proposed in [10] that a surrogate constraint, $\sum_i (y_i / upperbound\ of\ y_i) \leq s$, be used to handle the cardinality constraint. Note that there is no upper bound constraint in problem (Q_s) in this thesis.

Problem (Q_s) also arises in the subset selection problem in regression. In a traditional subset selection problem, m sets of sample data are given as (x_i, y_i), where $x_i \in R^n$ and $y_i \in R$, and the target is to find the best linear function $y = \beta' x$ under the L_2 criterion, such as $\min \sum_{i=1}^{m} (y_i - \beta' x_i)^2$. A closed form solution $\beta = (X'X)^{-1}X'Y$ has been derived for this problem, where $X \in R^{m \times n}$ with x_i' as its i-th row and $Y \in R^m$ with y_i as its i-th component. However, for adding robustness, it is often considered, as in [12], to be more appropriate to find a small subset of the variables instead of using all of the data for the regression.

$$minimize \quad (Y - X\beta)'(Y - X\beta)$$

$$\text{Subject to :} \quad |supp(\beta)| \leq k. \quad k < m.$$

This model is actually the same as (Q_s) if we ignore the constant part in the objective function. Currently, simple greedy heuristics [12], such as forward and backward regression, are the most commonly used solution methods to this problem, because they are fast, even for large-sized problems. However, these heuristic methods cannot guarantee an exact solution.

The rest of this thesis is organized as follows. In Chapter 2, a solution scheme is developed using dynamic programming, although this method is only practically useful for cases with a scalar state. In Chapter 3, problem (P_s) is treated as a cardinality constrained quadratic optimization problem (Q_s). First, the properties of this problem are studied and then an efficient branch and bound method is proposed to solve it. An illustrative example is presented to demonstrate the solution procedure. The results of numerical experiments are given to compare the computational performance between the method in this thesis and the CPLEX 9.1 solver, which is a commercial optimization software. The thesis concludes in Chapter 4 with some suggestions for future work.

Chapter 2

Solution Framework Using Dynamic Programming

In Section 1 of this chapter, the difficulties of using dynamic programming as a solution scheme for the general case of problem (P_s) is illustrated. In Section 2, however, it is shown that if state x is a scalar state, then dynamic programming provides an elegant solution scheme. In Section 3, the time-invariant system is discussed as a special case, and the computational procedure for an illustrative example problem is demonstrated in Section 4.

2.1 Difficulty of using dynamic programming

Dynamic programming is, without a doubt, the most powerful general methodology for optimal control problems that are separable with respect to time. Problem (P_s) becomes separable when the state space is expanded by adding an integer-valued variable r_t, which represents the remaining times of control implementation time at stage t. Obviously, r_t satisfies the recursive equation

$$\begin{aligned} r_0 &= s \\ r_{t+1} &= r_t - Supp(u_t). \end{aligned} \tag{2.1}$$

Define the cost-to-go at stage t for a given state x_t and r_t as

$$J_t(x_t, r_t) = \min_{u_t, \cdots, u_{T-1}} \{\sum_{k=t}^{T} x_k' Q_k x_k + \sum_{k=t}^{T-1} u_k' R_k u_k \mid x_t, r_t\}.$$

The following is evident from Lemma 1.1,

$$J_t(x_t, r_1) \leq J_t(x_t, r_2), \quad if \quad r_1 > r_2.$$

The interpretation of this inequality is that the existence of more control opportunities in time interval $[t, T]$ with the same starting point x_t leads to a better outcome.

To verify this, a simple example can first be examined. Assume that $T = 5$ and $s = 2$. At stage 5, the optimal r_5 can only be 0. At stage 4, the potential optimal r_4 can only belong to $\{0, 1\}$. Otherwise, if $r_t > 1$, then the remaining control will not be used up in stage 5. For the same reason, at stage 3 the feasible set of r_3 that is be considered is $\{0, 1, 2\}$. Figure 2.1 gives all of the possible situations in which optimality can be achieved and the feasible sets of r_t for $t = 0, 1, \ldots, 5$.

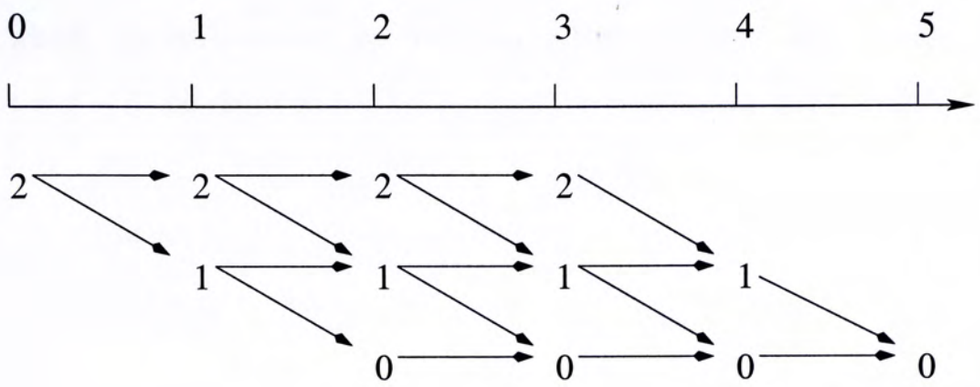


Figure 2.1: Potential Optimal Sets of r_t with T = 5 and s = 2

In general, the feasible sets of r_t to be considered for $t = 0, 1...T$ can be expressed as

$$F_t = \{n \in Z^+ \mid \max(0, s-t) \leq n \leq \min(s, T-t)\}. \tag{2.2}$$

Dynamic programming starts the solution process at stage $T-1$. The feasible set of r_t is $F_{T-1} = \{0, 1\}$.

For $r_{T-1} = 0$,

$$J_{T-1}(x_{T-1}, 0) = x'_{T-1} P^0_{T-1} x_{T-1},$$

where

$$P^0_{T-1} = Q_{T-1} + A'_{T-1} Q_T A_{T-1}.$$

For $r_{T-1} = 1$,

$$J_{T-1}(x_{T-1}, 1) = x'_{T-1} P^1_{T-1} x_{T-1},$$

where

$$P^1_{T-1} = A'_{T-1}[Q_T - Q_T B_{T-1}(B'_{T-1} Q_T B_{T-1} + R_{T-1})^{-1} B'_{T-1} Q_T] A_{T-1} + Q_{T-1},$$

which can be obtained by minimizing the cost-to-go with respect to u_{t-1}. At $t = T-1$, no decision has to be made on whether or not to implement control.

At stage $T-2$, the feasible set of r_{T-2} is $F_{T-2} = \{0, 1, 2\}$. Check the situation for $r_{T-2} = 1$.

$$\begin{aligned} J_{T-2}(x_{T-2}, 1) = \quad & \min\{x'_{T-2}Q_{T-2}x_{T-2} + J_{T-1}(A_{T-2}x_{T-2}, 1), \\ & \min_{u_{T-2}}[x'_{T-2}Q_{T-2}x_{T-2} + u'_{T-2}R_{T-2}u_{T-2} \\ & + J_{T-1}(A_{T-2}x_{T-2} + B_{T-2}u_{T-2}, 0)]\} \\ = \quad & \min\{x'_{T-2}\overline{P^1_{T-2}}x_{T-2}\ ,\ x'_{T-2}\widetilde{P^1_{T-2}}x_{T-2}\}, \end{aligned}$$

where

$$\begin{aligned} \widetilde{P^1_{T-2}} &= A'_{T-2}[P^0_{T-1} - P^0_{T-1}B_{T-2}(B'_{T-2}P^0_{T-1}B_{T-2} + R_{T-2})^{-1}B'_{T-2}P^0_{T-1}]A_{T-2} + Q_{T-2}, \\ \overline{P^1_{T-2}} &= A'_{T-2}P^1_{T-1}A_{T-2} + Q_{T-2}. \end{aligned}$$

Thus, for $r_{T-2} = 1$ at stage $T-2$, whether $Supp(u_{T-2}) = 0$ or 1 depends on whether

$$\begin{aligned} & x'_{T-2}\overline{P^1_{T-2}}x_{T-2} - x'_{T-2}\widetilde{P^1_{T-2}}x_{T-2} \\ & = x'_{T-2}(\overline{P^1_{T-2}} - \widetilde{P^1_{T-2}})x_{T-2} \geq 0 \end{aligned}$$

or not. Note that matrix $\overline{P^1_{T-2}} - \widetilde{P^1_{T-2}}$ may be indefinite, as can be seen by observing the following simple example.

$$A_{T-1} = A_{T-2} = Q_T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad B_{T-1} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \quad B_{T-2} = \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix},$$

$$R_{T-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad R_{T-2} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}.$$

Then,

$$\overline{P^1_{T-2}} - \widetilde{P^1_{T-2}} = \begin{bmatrix} -1.45 & 0 \\ 0 & 0.4 \end{bmatrix}.$$

In other words, whether $x'_{T-2}(\overline{P^1_{T-2}} - \widetilde{P^1_{T-2}})x_{T-2}$ is positive depends on state x_{T-2} when the dimension of the state is greater than 1.

Assume that at stage $t+1$ the optimal cost-to-go can be expressed as

$$J_{t+1}(x_{t+1}, r) = x'_{t+1} P^r_{t+1}(x_{t+1}) x_{t+1},$$

where P^r_{t+1} may be dependent to x_{t+1}.

At stage t, the optimal cost-to-go is given by the following formula for $r_t \in F_t$,

$$\begin{aligned} J_t(x_t, r_t) = \quad & \min\{x'_t Q_t x_t + J_{t+1}(A_t x_t, r_t), \min_{u_t}[x'_t Q_t x_t + u'_t R_t u_t \\ & + J_{t+1}(A_t x_t + B_t u_t, r_t - 1]\}. \end{aligned}$$

Thus, it is often impossible to solve this recursive equation analytically when state x is not scalar. In other words, dynamic programming usually fails to solve the multi-dimensional cardinality constraint linear-quadratic control problem.

When state x is scalar, however, it is possible to obtain solution recursively using dynamic programming.

2.2 Scalar-state problems

In this section, the focus is on the situation in which $x \in R^1$. Problem (P_s) is restated as

$$(\hat{P}_s) \qquad J(s) = \min\{Q_T x_T^2 + \sum_{t=0}^{T-1}(Q_t x_t^2 + u_t' R_t u_t)\}$$
$$\text{Subject to : } x_{t+1} = a_t x_t + B_t u_t$$
$$\sum_{t=0}^{T-1} Supp(u_t) \leq s < T,$$

where $x_t \in R^1$, $u_t \in R^m$, $a_t \in R$, $B_t \in R^{1\times m}$, $Q_t \in R$, $R_t \in R^{m\times m}$, R_t is positive definite, and $Q_t \geq 0$ for all t.

A major result for $(\hat{P}_s)$ is as follows.

Theorem 2.1 *For the scalar-state problem $(\hat{P}_s)$, whether or not an optimal control u_t is implemented at time t is determined by the following formula for $t = T-1, T-2.., 1, 0$.*

$$Supp(u_t) = \begin{cases} 1 & if \quad P_t^r = \widetilde{P_t^r} \\ 0 & if \quad P_t^r = \overline{P_t^r}, \end{cases}$$

where

$$P_t^r = \begin{cases} min\{\widetilde{P_t^r}, \overline{P_t^r}\} & if\ t+r < T \\ \widetilde{P_t^r} & if\ t+r = T \\ \overline{P_t^r} & if\ r = 0 \end{cases}$$

with

$$\widetilde{P_t^r} = \frac{a_t^2 P_{t+1}^{r-1}}{1 + P_{t+1}^{r-1}(B_t R_t^{-1} B_t')} + Q_t$$
$$\overline{P_t^r} = a_t^2 P_{t+1}^r + Q_t,$$

where the initial condition is given by $P_T^0 = Q_T$. *Furthermore, the magnitude of* u_t *is given by*

$$u_t = \begin{cases} 0 & if \quad Supp(u_t) = 0 \\ -a_t P_{t+1}^{r-1}(B_t' P_t^{r-1} B_t + R_t)^{-1} B_t' x_t & if \quad Supp(u_t) = 1 \end{cases}$$

and the corresponding objective value is $v^* = P_0^s x_0^2$.

Proof. The cost-to-go at (x_t, r) can be defined as

$$J(x_t, r) = \min_{u_t, .. u_{T-1}} [\sum_{k=t}^{T} Q_k x_k^2 + \sum_{k=t}^{T-1} u_k' R_k u_k \mid x_t, r],$$

where $r \in F_t$, r is defined in (2.1) and F_t is defined in (2.2). Because the dynamics are separable, dynamic programming can be used to derive the recursive equation of the cost-to-go,

$$J(x_t, r) = \min_{u_t} [Q_t x_t^2 + u_t' R_t u_t + J(x_{t+1}, r - Supp(u_t))].$$

In the following, the induction method is used to prove the result. Notice that the initial condition at T is

$$J(x_T, 0) = Q_T x_T^2.$$

It will be proved that the cost-to-go takes a quadratic form for all of the stages $T-1$,...,1,0,

$$J(x_t, r) = P_t^r x_t^2.$$

The induction method starts at $k = T-1$. The feasible set of r at stage $T-1$ is $F_{T-1} = \{0, 1\}$.

For $r = 0$, the cost-to-go is

$$J(x_{T-1}, 0) = P_{T-1}^0 x_{T-1}^2,$$

where

$$P_{T-1}^0 = Q_{T-1} + a_{T-1}^2 Q_T.$$

For $r = 1$, the cost-to-go is

$$\begin{aligned} J_{T-1}(x_{T-1}, 1) &= \min_{u_{T-1}} \{Q_{T-1} x_{T-1}^2 + u'_{T-1} R_{T-1} u_{T-1} + \\ &\quad + Q_T (a_{T-1} x_{T-1} + B_{T-1} u_{T-1})^2\}. \end{aligned}$$

The minimization of $J_{T-1}(x_{T-1}, 1)$ with respect to u_{T-1} identifies the optimal control,

$$u_{T-1}^* = -a_{T-1} P_T^0 (B'_{T-1} Q_T B_{T-1} + R_{T-1})^{-1} B'_{T-1} x_{T-1}.$$

The substitution of u_{T-1}^* back to function $J_{T-1}(x_{T-1}, 1)$ yields the minimum value,

$$J_{T-1}(x_{T-1}, 1) = a_{T-1}^2 (Q_T - Q_T^2 B_{T-1} (B'_{T-1} Q_T B_{T-1} + R_{T-1})^{-1} B'_{T-1}) + Q_{T-1}.$$

Notice the following well-known identity with proper dimensions for M_1, M_2, M_3, M_4

$$(M_1 + M_2 M_4^{-1} M_3)^{-1} = M_1^{-1} + M_1^{-1} M_2 (M_4 - M_3 M_1^{-1} M_2)^{-1} M_3 M_1^{-1}.$$

Thus,

$$(B'_{T-1} Q_T B_{T-1} + R_{T-1})^{-1} = R_{T-1}^{-1} - \frac{Q_T R_{T-1}^{-1} B'_{T-1} B_{T-1} R_{T-1}^{-1}}{1 + Q_T B_{T-1} R_{T-1}^{-1} B'_{T-1}},$$

and $J_{T-1}(x_{T-1}, 1)$ can be rewritten as

$$J_{T-1}(x_{T-1}, 1) = P_{T-1}^1 x_{T-1}^2,$$

where

$$P_{T-1}^1 = \left(\frac{a_{T-1}^2 P_T^0}{1 + Q_T B_{T-1} R_{T-1}^{-1} B'_{T-1}} + Q_{T-1}\right).$$

The induction assumption holds true for $T-1$.

Assume that the induction assumption holds true at stage $k+1$. The optimal cost-to-go takes the following form for a given x_{k+1} and $r_{k+1}=j$,

$$J(x_{k+1},j) = P_{k+1}^{j}x_{k+1}^2,$$

where $j \in F_{k+1}$. For convenience, let

$$J_{k+1}(x_{k+1},j) = +\infty,$$

for $j \notin F_{k+1}$.

At stage k, the cost-to-go can be calculated as follows for each $r \in F_k$,

$$\begin{aligned} J(x_k,r) &= \min_{u_k}[Q_k x_k^2 + Supp(u_k)\cdot u_k' R_k u_k + J(x_{k+1}, r-Supp(u_k))] \\ &= \min\{Q_k x_k^2 + J(x_{k+1},r), \min_{u_k}[Q_k x_k^2 + u_k' R_k u_k + J(x_{k+1},r-1)]\}. \end{aligned}$$

There are three possible situations.

i) If $r=0$, then $r-1 \notin F_{k+1}$, and the cost-to-go is

$$\begin{aligned} J(x_k,r) &= \min\{Q_k x_k^2 + J(x_{k+1},r), +\infty\} \\ &= \overline{P_k^r} x_k^2, \end{aligned}$$

where

$$\overline{P_k^r} = a_k^2 P_{k+1}^r + Q_k. \tag{2.3}$$

In this situation, no control action can be taken either in the current stage or in the future.

ii) If $r = T - k$, then $r \notin F_{k+1}$, and the cost-to-go is

$$\begin{aligned} J(x_k, r) &= \min\{+\infty, \min_{u_k}[Q_k x_k^2 + u_k' R_k u_k + J(x_{k+1}, r-1)]\} \\ &= \widetilde{P_k^r} x_k^2, \end{aligned}$$

where

$$\widetilde{P_k^r} = \frac{a_k^2 P_{k+1}^{r-1}}{1 + P_{k+1}^{r-1} B_k R_k^{-1} B_k'} + Q_k \tag{2.4}$$

and the optimal control is

$$u_k^* = -a_k P_{k+1}^{r-1} (B_k' P_{k+1}^{r-1} B_k + R_k)^{-1} B_k' x_k.$$

In this situation, control action must be taken at every remaining stage, otherwise the resulting control policy will not be optimal.

iii) If $r < T - k$, then both $r \in F_{k+1}$ and $r - 1 \in F_{k+1}$, and the cost-to-go can be calculated by

$$J(x_k, r) = \min\{\overline{P_k^r}, \widetilde{P_k^r}\} x_k^2,$$

where $\overline{P_k^r}$ and $\widetilde{P_k^r}$ are given above in (2.3) and (2.4), respectively. Denote

$$P_k^r = \min\{\overline{P_k^r}, \widetilde{P_k^r}\}.$$

Control action should be taken only when $\overline{P_k^r} > \widetilde{P_k^r}$, and it can be concluded that the induction assumption holds true for stage k, which completes the proof.

By using this theorem, optimal control can be obtained by recursively calculating a two-dimensional recursive equation for $(\overline{P_k^r}, \widetilde{P_k^r})$ from stage T to stage 0. An illustrative example is given in Section 2.4.

2.3 Time-invariant system

In this section, the time-invariant case of problem $(\hat{P}_s)$ is considered, and an interesting result is obtained.

Corollary 2.3.1 *Consider the time-invariant case of* $(\hat{P}_s)$, *where* $a_t = a$, $B_t = B$, $Q_t = Q > 0$, *and* $R_t = R$. *The optimal control strategy must satisfy* $Supp(u_0) = 1$, $Supp(u_1) = 1, \ldots, Supp(u_{s-1}) = 1$, *and* $Supp(u_s) = 0, \ldots, Supp(u_T) = 0$.

Remark: The idea behind the proof is to use theorem (2.1) recursively to prove that the claim $P_t^r = \widetilde{P_t^r}$ holds for all $r \geq 1$ and $r \in F_t$, where $t = 0, 1, \ldots, T-2$. The direct result of this claim is that the optimal control strategy is to implement control in the first s stages. Take Figure 2.1 as an example. At stage 3, it must be proved that $P_3^1 = \widetilde{P_3^1}$, at stage 2, it must be proved that $P_2^2 = \widetilde{P_2^2}$ and $P_2^1 = \widetilde{P_2^1}$, and at stage 1, it must be proved that $P_1^2 = \widetilde{P_1^2}$ and $P_1^1 = \widetilde{P_1^1}$. Lastly, at stage 0, it must be proved that $P_0^2 = \widetilde{P_0^2}$.

Proof. Recognizing (2.3) and (2.4), the following fact is evident for all $r \geq 1$ and $r \in F_t$, $t = 0, 1, \ldots, T-2$,

$$\begin{aligned} & P_t^r = \widetilde{P_t^r} \\ \Leftrightarrow\ & \widetilde{P_t^t} < \overline{P_t^r} \\ \Leftrightarrow\ & P_{t+1}^r P_{t+1}^{r-1} C > P_{t+1}^{r-1} - P_{t+1}^r. \end{aligned} \tag{2.5}$$

where $C = BR^{-1}B'$ is a positive scalar.

The mathematical induction method starts at $t = T-2$, and the feasible region of r is $F_{T-2} = \{0, 1, 2\}$. For the case $r = 2$, it is obvious that $P_{T-2}^2 = \widetilde{P_{T-2}^2}$

holds due to Lemma 1.1. For the case $r = 1$, relationship (2.5) implies that proving $P^1_{T-2} = \widetilde{P^1_{T-2}}$ is equivalent to proving

$$P^1_{T-1}P^0_{T-1}C > (P^0_{T-1} - P^1_{T-1}). \tag{2.6}$$

Using (2.3), (2.4), and $P^0_T = Q$ yields

$$\begin{aligned} & P^1_{T-1}P^0_{T-1}C - (P^0_{T-1} - P^1_{T-1}) \\ &= C(\frac{a^2Q}{1+QC} + Q)(a^2Q + Q) - C(\frac{a^2Q^2}{1+QC}) > 0, \end{aligned}$$

which completes the proof for $t = T - 2$.

Assume that $P^r_t = \widetilde{P^r_t}$ for all $r \geq 1$ and $r \in F_t$, where $t = l+1, l+2, \ldots, T-2$. The next target is to prove that for any $r \geq 1$ and $r \in F_l$, $P^r_l = \widetilde{P^r_l}$, or equivalently,

$$P^r_{l+1}P^{r-1}_{l+1}C > P^{r-1}_{l+1} - P^r_{l+1}, \tag{2.7}$$

holds true.

(i) For $r = 1$, Theorem 2.1 and the induction assumption are used to rewrite P^1_{l+1} and P^0_{l+1} as

$$\begin{aligned} P^1_{l+1} &= \frac{a^2P^0_{l+2}}{1+CP^0_{l+2}} + Q, \\ P^0_{l+1} &= a^2P^0_{l+2} + Q. \end{aligned}$$

Thus,

$$\begin{aligned} & CP^1_{l+1}P^0_{l+1} - (P^0_{l+1} - P^1_{l+1}) \\ &= C(\frac{a^2P^0_{l+2}}{1+CP^0_{l+2}} + Q)(a^2P^0_{l+2} + Q) - C\frac{a^2(P^0_{l+2})^2}{1+CP^0_{l+2}}. \end{aligned} \tag{2.8}$$

Notice that

$$P_j^0 = Q\sum_{n=0}^{T-j} a^{2n}. \tag{2.9}$$

for any $j = 0, 1, \ldots, T-1$. It follows that

$$\begin{aligned} a^2 P_{l+2}^0 + Q &= a^{2(T-l-1)}Q + P_{l+2}^0 \\ \Rightarrow a^2 P_{l+2}^0 + Q &> P_{l+2}^0. \end{aligned}$$

Substituting this inequality into (2.8) gives $CP_{l+1}^1 P_{l+1}^0 > (P_{l+1}^0 - P_{l+1}^1)$ for $r = 1$.

(ii) For $r > 1$, the left-hand side of (2.7) is considered first. Because of the induction assumption and Theorem 2.1, P_{l+1}^r can be expressed as

$$P_{l+i}^{r+1-i} = \frac{a^2 P_{l+i+1}^{r-i}}{1 + CP_{l+i+1}^{r-i}} + Q. \tag{2.10}$$

Letting $i = 1, 2, \ldots, r$ in the aforementioned equation yields

$$\begin{aligned} P_{l+1}^r &= \frac{a^2 P_{l+2}^{r-1}}{1 + CP_{l+2}^{r-1}} + Q > \frac{a^2 P_{l+2}^{r-1} + Q}{1 + CP_{l+2}^{r-1}}, \\ P_{l+2}^{r-1} &= \frac{a^2 P_{l+3}^{r-2}}{1 + CP_{l+3}^{r-2}} + Q > \frac{a^2 P_{l+3}^{r-2} + Q}{1 + CP_{l+3}^{r-2}}, \\ &\vdots \\ P_{l+r}^1 &= \frac{a^2 P_{l+r+1}^0}{1 + CP_{l+r+1}^0} + Q > \frac{a^2 P_{l+r+1}^0 + Q}{1 + CP_{l+r+1}^0}. \end{aligned}$$

The performance of successive substitution and the dropping of the part $CP_{l-i+1}^{r-i}Q$ in the numerator for the iteration for i yields

$$P_{l+1}^r > \frac{a^{2r} P_{l+r+1}^0 + Q\sum_{i=0}^{r-1} a^{2i}}{\prod_{i=1}^{r}(1 + CP_{l+i+1}^{r-i})}. \tag{2.11}$$

Similarly, P_{l+1}^{r-1} can be expressed as

$$P_{l+i}^{r-i} = \frac{a^2 P_{l+i+1}^{r-i-1}}{1 + CP_{l+i+1}^{r-i-1}} + Q, \tag{2.12}$$

for $i = 1, 2, \ldots, r-1$. It follows that

$$P_{l+1}^{r-1} > \frac{a^{2(r-1)} P_{l+r}^{0} + Q \sum_{i=0}^{r-2} a^{2i}}{\prod_{i=1}^{r-1} (1 + C P_{l+i+1}^{r-i-1})}. \tag{2.13}$$

Solving (2.12) and (2.10) recursively provides the following expression for the right-hand side of (2.7),

$$P_{l+1}^{r-1} - P_{l+1}^{r} = \frac{a^{2(r-1)} (P_{l+r}^{0} - P_{l+r}^{1})}{\prod_{i=1}^{r-1} [(1 + C P_{l+i+1}^{r-i})(1 + C P_{l+i+1}^{r-i-1})]}.$$

Because of the induction assumption,

$$P_{l+r}^{0} = a^2 P_{l+r+1}^{0} + Q$$

and

$$P_{l+r}^{1} = \frac{a^2 P_{l+r+1}^{0}}{1 + C P_{l+r+1}^{0}} + Q.$$

Thus,

$$P_{l+1}^{r-1} - P_{l+1}^{r} = \frac{C a^{2r} (P_{l+r+1}^{0})^2}{[\prod_{i=1}^{r} (1 + C P_{l+i+1}^{r-i})][\prod_{i=1}^{r-1} (1 + C P_{l+i+1}^{r-i-1})]}. \tag{2.14}$$

From (2.9),

$$a^{2(r-1)} P_{l+r}^{0} + Q \sum_{i=0}^{r-2} a^{2i} = \sum_{i=T-l-r}^{T-l-1} a^{2i} Q + P_{l+r+1}^{0} > P_{l+r+1}^{0}.$$

Thus, from (2.11), (2.13), and (2.14), it can be concluded that (2.7) holds true for $r > 1$, and the proof is complete.

2.4 Illustrative example of a scalar-state problem

The following illustrative example for problem $(\hat{P}_s)$ is now considered.

$$
\begin{aligned}
(\hat{P}_3) \qquad & J(3) = \min\{\sum_{t=0}^{6} Q_t x_t^2 + \sum_{t=0}^{5} u_t' R_t u_t\} \\
& \text{s.t. } x_{t+1} = a_t x_t + B_t u_t \\
& \sum_{t=0}^{T-1} Supp(u_t) \leq 3,
\end{aligned}
$$

where the data are given as,

$a_0 = -1.0814, \quad a_1 = -4.1640, \quad a_2 = 0.3133,$

$a_3 = 0.7192, \quad a_4 = -2.8662, \quad a_5 = 2.9773;$

$$
\begin{aligned}
&B_0 = \begin{bmatrix} -0.4326 & 1.1892 & -0.5883 \end{bmatrix}; \; B_1 = \begin{bmatrix} -1.6656 & -0.0376 & 2.1832 \end{bmatrix}, \\
&B_2 = \begin{bmatrix} 0.1253 & 0.3273 & -0.1364 \end{bmatrix}, \; B_3 = \begin{bmatrix} 0.2877 & 0.1746 & 0.1139 \end{bmatrix}, \\
&B_4 = \begin{bmatrix} -1.1465 & -0.1867 & 1.0668 \end{bmatrix}, \; B_5 = \begin{bmatrix} 1.1909 & 0.7258 & 0.0593 \end{bmatrix};
\end{aligned}
$$

$Q_0 = 29.1231, \; Q_1 = 46.0058, \; Q_2 = 15.2920, \; Q_3 = 36.4065, \; Q_4 = 0.6004,$

$Q_5 = 9.4872, \; Q_6 = 20.4342,$

$$
R_0 = \begin{bmatrix} 55.2073 & -35.3202 & -69.8318 \\ -35.3202 & 63.5860 & 0.1241 \\ -69.8318 & 0.1241 & 275.3748 \end{bmatrix}, \; R_1 = \begin{bmatrix} 47.4482 & 1.2527 & 1.5143 \\ 1.2527 & 48.4869 & -0.2429 \\ 1.5143 & -0.2429 & 44.5971 \end{bmatrix},
$$

$$
R_2 = \begin{bmatrix} 418.1174 & -79.3356 & 0.8914 \\ -79.3356 & 381.4516 & -27.9630 \\ 0.8914 & -27.9630 & 456.8184 \end{bmatrix}, \; R_3 = \begin{bmatrix} 36.2151 & 1.0870 & 0.5769 \\ 1.0870 & 32.4531 & -0.3443 \\ 0.5769 & -0.3443 & 42.5195 \end{bmatrix},
$$

$$R_4 = \begin{bmatrix} 377.9890 & -63.7029 & -7.2765 \\ -63.7029 & 352.0914 & -1.8472 \\ -7.2765 & -1.8472 & 438.2543 \end{bmatrix}, R_5 = \begin{bmatrix} 31.2435 & 6.8929 & -2.5035 \\ 6.8929 & 22.2387 & 0.3656 \\ -2.5035 & 0.3656 & 18.9408 \end{bmatrix}.$$

Table 2.1 shows the procedure for the solution of the example problem. More details of the solution process are given in the following.

Stage 6: for $r = 0$, $P_6^0 = 20.4$;

Stage 5: for $r = 0$, $P_5^0 = 190.6$;

for $r = 1$, $P_5^1 = 93.4$;

Stage 4: for $r = 0$, $P_4^0 = 1566.6$;

for $r = 1$, $\overline{P_4^1} = 767.8$, $\widetilde{P_4^1} = 707.5$, thus $P_4^1 = 707.4908$;

for $r = 2$, $P_4^2 = 481.5$;

Stage 3: for $r = 0$, $P_3^0 = 846.7$;

for $r = 1$, $\overline{P_3^1} = 402.4$, $\widetilde{P_3^1} = 163.7$, thus $P_3^1 = 163.7$;

for $r = 2$, $\overline{P_3^2} = 285.5$, $\widetilde{P_3^2} = 143.3$, thus $P_3^2 = 143.3$;

for $r = 3$, $P_3^3 = 130.4$

Stage 2: for $r = 1$, $\overline{P_2^1} = 31.4$, $\widetilde{P_2^1} = 77.4$, thus $P_2^1 = 31.4$;

for $r = 2$, $\overline{P_2^2} = 29.4$, $\widetilde{P_2^2} = 30.4$, thus $P_2^2 = 29.4$;

for $r = 3$, $\overline{P_2^3} = 28.1$, $\widetilde{P_2^3} = 28.6$, thus $P_2^3 = 28.1$;

Stage 1: for $r = 2$, $\overline{P_1^2} = 555.0$, $\widetilde{P_1^2} = 131.6$, thus $P_1^2 = 131.6$;

for $r = 3$, $\overline{P_1^3} = 533.1$, $\widetilde{P_1^3} = 130.7$, thus $P_1^3 = 130.7$;

Stage 0: for $r = 3$, $\overline{P_0^3} = 181.9$, $\widetilde{P_0^3} = 66.3$, thus $P_0^3 = 66.3$.

After this backward calculation, the direction is reversed to fix $Supp(u_t)$:

$Supp(u_0) = 1$; $Supp(u_1) = 1$;

$Supp(u_2) = 0$; $Supp(u_3) = 1$;

$Supp(u_4) = 0$; $Supp(u_5) = 0$.

At the same time, the optimal control can be found in the stages in which $Supp(u_t) = 1$,

$$u_0 = \begin{bmatrix} -0.1504 \\ -0.7267 \\ 0.0357 \end{bmatrix} x_0, \quad u_1 = \begin{bmatrix} 0.7545 \\ -0.0087 \\ -1.0316 \end{bmatrix} x_1, \quad u_3 = \begin{bmatrix} 1.3711 \\ 0.9110 \\ 0.4628 \end{bmatrix} x_3.$$

The optimality of this strategy can be verified by enumerating all of the possible situations in Table 2.2.

Table 2.1: Solution procedure of the algorithm

No. step	t	F_t	P_t^m	The computational process of P_t^m
1	6	$\{0\}$	P_6^0	$P_6^0 = Q_6$
2	5	$\{0,1\}$	P_5^0, P_5^1	$P_5^0 \leftarrow P_6^0, P_5^1 \leftarrow P_6^0$
3	4	$\{0,1,2\}$	P_4^0, P_4^1, P_4^2	$P_4^0 \leftarrow P_5^0, P_4^1 \leftarrow P_5^0, P_4^2 \leftarrow P_5^1$
4	3	$\{0,1,2,3\}$	$P_3^0, P_3^1, P_3^2, P_3^3$	$P_3^0 \leftarrow P_4^0, P_3^1 \leftarrow P_4^0, P_3^2 \leftarrow P_4^1, P_3^3 \leftarrow P_4^2$
5	2	$\{1,2,3\}$	P_2^1, P_2^2, P_2^3	$P_2^1 \leftarrow P_3^1, P_2^2 \leftarrow P_3^2, P_2^3 \leftarrow P_3^3$
6	1	$\{2,3\}$	P_1^3, P_1^2	$P_1^2, \leftarrow P_2^1, P_1^3 \leftarrow P_2^2$
7	0	$\{3\}$	P_0^3	$P_0^3 \leftarrow P_1^2$

Table 2.2: Result of the complete enumeration

$Supp(u_t)$ $t = 0, 1, \ldots, 5$	Optimal value
1 1 1 0 0 0	66.8805
* 1 1 0 1 0 0	66.3064
1 0 1 1 0 0	74.8100
0 1 1 1 0 0	182.4791
1 1 0 0 1 0	66.7135
1 0 1 0 1 0	75.9750
1 0 0 1 1 0	74.7116
0 1 1 0 1 0	189.0695
0 1 0 1 1 0	181.9271
0 0 1 1 1 0	662.7013
0 1 0 1 0 1	182.0644
0 1 0 0 1 .1	187.5329
1 1 0 0 0 1	66.7435
1 0 1 0 0 1	76.0616
1 0 0 1 0 1	74.7361
1 0 0 0 1 1	75.7050
0 1 1 0 0 1	189.5634
0 0 1 1 0 1	667.1678
0 0 1 0 1 1	902.8433
0 0 0 1 1 1	652.5359

Chapter 3

Cardinality Constrained Quadratic Optimization

In this chapter, problem (P_s) is reformulated as a special case of the cardinality constrained quadratic optimization problem (Q_s). In Section 1, the reformulation of (P_s) to (Q_s) is given, and it is briefly demonstrated in Section 2 that problem (Q_s) is an NP-hard problem. In Section 3, an efficient branch and bound method is introduced to solve the problem, and in the last section, an illustrative example and the computational performance are presented.

3.1 Reformulation

For problem (P_s), the linearity of the systems dynamic implies that the state x_t can be written as a function of u_τ, $\tau = 0, 1, \ldots, t-1$, at any stage $t \geq 1$,

$$x_t = \prod_{i=0}^{t-1} A_i x_0 + \sum_{\tau=0}^{t-1} [\prod_{j=\tau+1}^{t-1} A_j B_\tau u_\tau],$$

where $\prod_{i=0}^{t-1} A_i = A_{t-1} \ldots A_0$ and $\prod_{i=t}^{t-1} A_i = 1$. Thus, (P_s) can be reformulated as a static optimization problem. To give a systematic and compact method for the reformulation, the following fictitious outputs are introduced.

As Q_t and R_t are positive definite, there exit $\hat{Q}_t = Q_t^{\frac{1}{2}}$ and $\hat{R}_t = R_t^{\frac{1}{2}}$ such that $Q_t = \hat{Q}_t \hat{Q}_t$ and $R_t = \hat{R}_t \hat{R}_t$. Define

$$C_t = \begin{bmatrix} \hat{Q}_t \\ 0 \end{bmatrix}, \quad D_t = \begin{bmatrix} 0 \\ \hat{R}_t \end{bmatrix},$$

where C_t is an $(m+n) \times n$ matrix and D_t is an $(m+n) \times m$ matrix. The fictitious outputs,

$$\begin{aligned} y_t &= C_t x_t + D_t u_t \quad for\ t = 1, 2, \ldots, T-1 \\ y_T &= \hat{Q}_T x_T \end{aligned}$$

are introduced. Notice that $C_t' D_t = 0$, and that the objective function of (P_s) can therefore be expressed as

$$\begin{aligned} J &= y_T' y_T + \sum_{t=0}^{T-1} y_t' y_t \\ &= \|Y_T\|^2, \end{aligned}$$

where $Y_T = [y_0\ y_1\ \cdots\ y_T]'$. The following matrices are then defined.

$$U = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{T-1} \end{bmatrix}, \qquad L = \begin{bmatrix} C_0 \\ C_1A_0 \\ \vdots \\ C_T\prod_{i=1}^{T-1} A_i \end{bmatrix},$$

$$M_t = \begin{bmatrix} D_0 & 0 & 0 & \cdots & 0 \\ C_1B_0 & D_1 & 0 & \cdots & 0 \\ C_2A_1B_0 & C_2B_1 & D_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & 0 \\ C_{t-1}(\prod_{i=1}^{t-2} A_i)B_0 & C_{t-1}(\prod_{i=2}^{t-3} A_i)B_1 & C_{t-1}(\prod_{i=3}^{t-4} A_i)B_2 & \cdots & D_{t-1} \end{bmatrix},$$

and

$$N_t = \begin{bmatrix} C_t(\prod_{i=1}^{t-1} A_i)B_0 & C_t(\prod_{i=2}^{t-1} A_i)B_1 & C_t(\prod_{i=3}^{t-1} A_i)B_2 & \cdots & C_tB_{t-1}. \end{bmatrix}.$$

Thus, a compact form of the fictitious output can be expressed as

$$Y_T = Lx_0 + MU,$$

where

$$M = \begin{bmatrix} M_T \\ N_T \end{bmatrix}. \tag{3.1}$$

The size of M_T is $(m+n)T \times m(T-1)$, the size of N_T is $2(m+n) \times m(T-1)$, and the size of matrix M is $(m+n)T \times m(T-1)$. The original problem (P_s) now becomes

$$(CP_s) \quad \min \quad J = \frac{1}{2}U'GU + g'U + constant$$
$$s.t \quad \sum_{t=0}^{T-1}(Supp(u_t)) \leq s,$$

where

$$G = 2M'M \qquad g = 2ML.$$

Matrix G is a $(Tm) \times (Tm)$ matrix and g is a $(Tm) \times 1$ vector.

There is a slight difference between problem (Q_s) and (CP_s), if $u_t \in R^m$ and $m > 1$. However, the following modification can be adopted.

$$\begin{aligned} (CP_s) \quad & \min \quad J = \frac{1}{2}U'ZGZU + b'ZU \\ & s.t \quad \sum_{t=0}^{T-1}(Supp(u_t)) \leq s, \end{aligned}$$

where

$$Z = \begin{bmatrix} I_{m\times m} \cdot Supp(u_0) & 0 & \cdots & 0 \\ 0 & I_{m\times m} \cdot Supp(u_1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & I_{m\times m} \cdot Supp(u_{T-1}) \end{bmatrix}.$$

In other words, the $m \times m$ elements that are associated with each u_t can be bundled together. Problem (CP_s) and (Q_s) are of the same formulation, if each $m \times m$ block in problem (CP_s) is treated as an element in problem (Q_s).

It is still necessary to demonstrate that $G = M'M$ is positive definite in problem (CP_s). If MU is regarded as a vector, then $(MU)'MU$ is simply the norm of this vector, which means that $U'M'MU > 0$, if $MU \neq 0$. The definition of D_t implies that the rank of D_t is m. It follows that the rank of M is mT, which means that, all of the columns of M are independent, and it can thus be concluded that $MU = 0$ implies $U = 0$. That is to say, $U'GU > 0$ for any $U \neq 0$.

By the definition of the positive definite matrix, it can concluded that $G \succ 0$ in the reformulated problem (CP_s).

3.2 NP hardness

In this section, it is shown that problem (Q_s) is an inherent NP-hard problem. Although the techniques that are used to prove NP-completeness vary almost as widely as the NP-complete problems themselves, there are some widely adopted general methods. The *restriction* method is the most frequently used technique (see [5]). To prove that a given problem $\Pi \in NP$ is NP-complete using this technique requires the identification of a known NP-complete problem Π' as a special case of Π. Problem Π' is called the restriction of problem Π, which is restated in the following lemma.

Lemma 3.1 *Problem* Π *is an NP-complete if problem* Π' *is NP-complete and* Π' *is the restriction of* Π.

The gist of such a technique lies in the transformation of a known NP-complete problem to the target problem by the addition of further restrictions. Here, only the one-to-one correspondence between the instances of the restricted problem and the known NP-complete problem is required. In the following part, some known NP-complete problems are given first, which is followed by the proof that problem (Q_s) is NP-hard.

Take the following well-known NP-complete problem.

•SUBSET SUM:

INSTANCE: A finite set A, with a "size" $s(a) \in Z^+$ for each $a \in A$ and a positive integer B.

QUESTION: Is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = B$?

This problem belongs to the class of NP-completeness problems that are listed in [5]. The following problem is closely related to the SUBSET SUM problem,

- **k-SUBSET SUM:**

INSTANCE: A finite set A, with a "size" $s(a) \in Z^+$ for each $a \in A$ and positive integers k and B.

QUESTION: Is there a subset $A' \subseteq A$ such that $|A'| = k$ and $\sum_{a \in A'} s(a) = B$?

The NP-hardness of this problem is obvious, because if the k-SUBSET SUM problem can be solved by polynomial time, then SUBSET SUM can also be solved by polynomial time, which contradicts the NP-completeness of the SUBSET SUM problem.

Another decision problem is now introduced. In the following problem, the notation of vector $x > 0$ implies that all of the elements of this vector are positive.

- **MATRIX PARTITION:**

INSTANCE: Given n vectors $m_1, m_2, \cdots, m_n \in R^p$ such that $m_i > 0, i = 1, 2, ..., n-1$ and $m_n < 0$, a positive integer k, and a matrix $M = (m_1, m_2, \ldots, m_n)$.

QUESTION: Is there a vector $x \in \{0,1\}^n$ such that $x'e = k$ and $Mx = 0$?

Lemma 3.2 *MATRIX PARTITION is NP-complete.*

Proof. The situation is first confined to $p = 1$. Let set $A = \{m_1, m_2, \cdots, m_{n-1}\}$ and regard $-m_n$ as a positive number. The problem becomes that of choosing subset $A' \subseteq A$ such that $|A'| = k$ and $\sum_{i \in A'} m_i = -m_n$, which is indeed a

K-SUBSET SUM problem. MATRIX PARTITION is NP-complete by the restriction method.

Theorem 3.1 *Problem (Q_s) is NP-hard.*

Proof. Construct a special case of problem (Q_s) as its restriction,

$$(\overline{Q}_s) \quad \overline{J} = \min \ \frac{1}{2}x'M'Mx$$
$$Supp(x) = s,$$

where $Supp(x)$ is the number of components that are not equal to 0. The matrix M is constructed as $M = (m_1, m_2, \ldots, m_n)$, where $m_i \in R^p$, $m_i > 0$ for all $i = 1, 2, \ldots, n-1$ and $m_n < 0$. It is obvious that $\overline{J} \geq 0$. Solving such an optimization problem is at least as difficult as asking the decision question I, "Is the optimal value of $(\overline{Q_s})$ equal to 0? ". (see [6]). If it is claimed that there is a polynomial time algorithm to solve problem $(\overline{Q_s})$, then there must be a polynomial time algorithm to answer question I. It follows that MATRIX PARTITION can be solved using a polynomial time algorithm, which contradicts the fact that MATRIX PARTITION is an NP-complete problem.

This theorem reveals that problem (Q_s) is an NP-hard problem and it is widely believed that no polynomial time algorithm can solve this class of problems.

3.3 Solving CCQP with an efficient branch and bound method

3.3.1 Efficient branch and bound algorithm

When $s = S$, the cardinality constrained optimization problem (Q_s) reduces to the conventional unconstrained convex quadratic optimization problem, the analytical solution and correspondent optimal value of which are well known to be $y^* = -G^{-1}g$ and $v^* = -\frac{1}{2}g'G^{-1}g$ respectively. When $s < S$, problem (Q_s) becomes non-convex, mainly because of its non-convex feasible region. Consider a simple case in which $S = 2$ and $s = 1$. It is easy to see that the non-convex feasible region for this simple case is a union of the two axes in the y-space. Thus, it is very difficult to develop the optimality condition for the global minimum. Furthermore, problem (Q_s) is of a combinatorial nature. Solving problem (Q_s) can be accomplished by comparing the optimal values of the C_S^s s-dimensional unconstrained convex quadratic optimization problem, where $C_S^s = \frac{S!}{s!(S-s)!}$ is the number of s unordered outcomes that can be chosen from S possibilities. Obviously, it is impossible to enumerate all of the possible situations, even for a moderate-sized problem with $S = 50$ and $s = 20$.

Define the index set for problem (Q_s) as follows, where $1 \leq s \leq S$.

$$I_s^{[j]} = \{j_1, j_2, \ldots, j_s \mid \{j_1, j_2, \ldots, j_s\} \subseteq I_S\},$$

where

$$I_S = \{1, 2, \ldots, S\}.$$

This index set can be interpreted as the selection of s indices from S total indices.

Denote the complementary set of $I_s^{[j]}$ as

$$\overline{I_s^{[j]}} \;=\; I_S \setminus I_s^{[j]}.$$

Without loss of generality, it can be assumed that $j_1 < j_2 < \ldots < j_s$. Clearly, there are C_S^s possible index sets of $I_s^{[j]}$ for problem (Q_s). Consider the following reduced convex quadratic optimization problem.

$$\begin{aligned}(Q_s^{[j]}) \quad & \min_y \frac{1}{2} y'Gy + g'y \\ & \text{subject to} \quad y_i = 0, \ \forall i \in \overline{I_s^{[j]}}.\end{aligned}$$

It is clear that the solution value of y_i, $i \in I_s^{[j]}$, in problem $(Q_s^{[j]})$ can be obtained by solving the formulation,

$$\min_{y_s^{[j]}} \frac{1}{2}(y_s^{[j]})'G_s^{[j]}y_s^{[j]} + (g_s^{[j]})'y_s^{[j]},$$

where $y_s^{[j]} = [y_{j_1}, y_{j_2}, \ldots, y_{j_s}]'$, $g_s^{[j]} = [g_{j_1}, g_{j_2}, \ldots, g_{j_s}]'$, and $G_s^{[j]}$ is formed by taking rows j_1, j_2, ..., j_s and columns j_1, j_2, ..., j_s from G. It is well known that any principle sub-matrix of a positive definite matrix is positive definite, which makes it a well-posed problem.

In the following, the optimal value of problem (Q_s) and problem $(Q_s^{[j]})$ are defined as $v(Q_s)$ and $v(Q_s^{[j]})$, respectively. Thus, the following is obvious,

$$v(Q_s) = \min_{I_s^{[j]}} v(Q_s^{[j]}).$$

Definition 1 *$I_s^{[j^s]}$ is said to be the optimal index set of Q_s if $v(Q_s) = v(Q_s^{[j^s]})$.*

When $I_s^{[j]}$ is an optimal index set of (Q_s), any solution to $(Q_s^{[j]})$ is an optimal solution to (Q_s).

Theorem 3.2 *Assume that $s < k$. If there are one index set $I_k^{[j^k]}$ for problem (Q_k) and one index set $I_s^{[j^s]}$ for problem (Q_s) such that $v(Q_k^{[j^k]}) > v(Q_s^{[j^s]})$, then no subset $I_s^{[j]} \subset I_k^{[j^k]}$ will be an optimal index set for problem Q_s.*

Proof. As for any $I_s^{[j]} \subset I_k^{[j^k]}$, $v(Q_s^{[j]})) \geq v(Q_k^{[j^k]})$, and further

$$v(Q_s^{[j]}) \geq v(Q_k^{[j^k]}) > v(Q_s^{[j^s]}) \tag{3.2}$$

for any $I_s^{[j]} \subset I_k^{[j^k]}$. The conclusion of the theorem follows from the definition of the optimal index set.

Let $I_s^{[j^*]}$ be the incumbent index set for problem (Q_s) and v_s^* be the value of $v(Q_s^{[j^*]})$. The following two corollaries can be easily obtained from Theorem 3.2.

Corollary 3.3.1 *Assume that $k > s$. If for an index set $I_k^{[\tilde{j}]}$, $v(Q_k^{[\tilde{j}]}) > v_s^*$, then at least one element from $\overline{I_k^{[\tilde{j}]}}$ must be in the optimal index set for problem (Q_s).*

Corollary 3.3.2 *If for an index set $I_{S-1}^{[\tilde{j}]}$, $v(Q_{S-1}^{[\tilde{j}]}) > v_s^*$, then the element of $\overline{I_{S-1}^{[\tilde{j}]}}$ must be in the optimal index set for problem (Q_s).*

Example 3.1 *The proposed solution concept is now illustrated by considering the following example (Q_4) with $S = 7$ and $s = 4$, where the data are given as*

$$G = \begin{bmatrix} 11.58 & -0.08 & 0.24 & -0.02 & -0.13 & -0.00 & -0.05 \\ -0.08 & 17.38 & -3.04 & 0.26 & 1.58 & 0.23 & 0.26 \\ 0.24 & -3.04 & 28.29 & 0.18 & -8.73 & 0.20 & -2.64 \\ -0.02 & 0.26 & 0.18 & 15.97 & 1.85 & -1.02 & 1.21 \\ -0.13 & 1.58 & -8.73 & 1.85 & 17.62 & -0.36 & -0.02 \\ -0.00 & 0.23 & 0.20 & -1.02 & -0.36 & 16.89 & 4.65 \\ -0.05 & 0.26 & -2.64 & 1.21 & -0.02 & 4.65 & 18.09 \end{bmatrix},$$

$$g' = \begin{bmatrix} 175.50 & 141.63 & 195.05 & 183.66 & 189.59 & 125.53 & 152.11 \end{bmatrix}.$$

The following can be verified

$$\begin{aligned}
I_6^{[1]} &= \{2,3,4,5,6,7\}, \quad v(Q_6^{[1]}) = -5236.14;\\
I_6^{[2]} &= \{1,3,4,5,6,7\}, \quad v(Q_6^{[2]}) = -5914.73;\\
I_6^{[3]} &= \{1,2,4,5,6,7\}, \quad v(Q_6^{[3]}) = -4537.07;\\
I_6^{[4]} &= \{1,2,3,5,6,7\}, \quad v(Q_6^{[4]}) = -5920.66;\\
I_6^{[5]} &= \{1,2,3,4,6,7\}, \quad v(Q_6^{[5]}) = -4740.22;\\
I_6^{[6]} &= \{1,2,3,4,5,7\}, \quad v(Q_6^{[6]}) = -6307.91;\\
I_6^{[7]} &= \{1,2,3,4,5,6\}, \quad v(Q_6^{[7]}) = -6021.86.
\end{aligned}$$

A natural conjecture from Corollary 3.3.2 is that the larger the value of $v(Q_6^{[j]})$, *the more important the variable* $\overline{I_6^{[j]}}$. *Thus, the variables in* y *can be ranked in the following way according to the decreasing order of their corresponding values of* $v(Q_6^{[j]})$,

$$\{3,5,1,2,4,7,6\}.$$

The first four members in this ranking order are used first in this cardinality constrained convex quadratic programming problem as a solution candidate. Let

$$I_4^{[j_1]} = \{3,5,1,2\}.$$

Solving $Q_4^{[j_1]}$ *yields an incumbent* $(y_1, y_2, y_3, y_5)' = (-15.1, -9.0, -12.8, -16.4)'$ *and its corresponding objective* $v_4^* = v(Q_4^{[j^1]}) = -4766.16$.

As $v(Q_6^{[3]}) = -4537.07 > v_4^*$ *and* $v(Q_6^{[5]}) = -4740.22 > v_4^*$, $\{3,5\} = \{\overline{I_6^{[3]}}, \overline{I_5^{[5]}}\}$ *must be in the optimal index set of* (Q_4) *according to Corollary 3.3.2.*

Let

$$I_5^{[j_1]} = \{3,4,5,6,7\}.$$

Solving $Q_5^{[j_1]}$ *gives* $v(Q_5^{[j_1]}) = -4589.44$, *which is larger than* $v_4^* = -4766.16$. *Thus, at least one of* $\{1,2\} = \overline{I_5^{[j_1]}}$ *must be in the optimal index set of* (Q_4) *according to Corollary 3.3.1.*

Let

$$I_5^{[j_2]} = \{2,3,5,6,7\},$$

$$I_5^{[j_3]} = \{2,3,4,5,6\}.$$

Both $v(Q_5^{[j_2]}) = -4589.46$ *and* $v(Q_5^{[j_3]}) = -4692.93$ *are greater than* $v_4^* = -4766.16$, *and thus at least one of* $\{1,4\} = \overline{I_5^{[j_2]}}$ *and at least one of* $\{1,7\} = \overline{I_5^{[j_3]}}$ *must be in the optimal index set of* (Q_4) *according to Corollary 3.3.1.*

Notice that two members in the optimal index set, $\{3,5\}$, *have been determined, and therefore not all of* $\{2,4,7\}$ *can be members of the optimal index set at the same time. This implies that* $\{1\}$ *must be in the optimal index set.*

Three members in the optimal index set, $\{1,3,5\}$, *have now been confirmed. The remaining single member in the optimal index set can be found out by checking* $Q_4^{[j]}$ *for* $I_4^{[j]} = \{1,3,4,5\}$, $\{1,3,5,6\}$, *and* $\{1,3,5,7\}$, *which identifies the optimal index set* $\{1,3,5,7\}$ *and an optimal solution to* (Q_4), $(y_1, y_3, y_5, y_7) = (-15.1, -13.1, -17.4, -10.4)$ *with* $v(Q_4) = -5040.68$.

Note that the total enumeration requires the evaluation of $C_7^4 = 35$ possible index sets by solving the corresponding unconstrained convex quadratic optimization problems of (Q_4). Using the optimality conditions that are given in Theorem 1, Corollary 1, and Corollary 2, only 13 unconstrained convex quadratic optimization problems need to be carried out. This solution scheme provides a bounding condition that can be used to construct a branch and bound algorithm to solve problem (Q_s).

In the following, the structure of a branch and bound algorithm that is combined with the *depth-first searching strategy* for the enumeration tree that is to be used as an exact solution methodology is described.

A branching order table I_t is defined to store the branching order of the index. The notation $I_t(i)$ denotes the i-th element of I_t, where $i = 1, 2, \ldots, s$.

The following members store the information at every node in the enumeration tree.

- *pard*: The address of the parent node.
- The index set U: To record which decision variables have been branched up, where branch up means that the variable takes a non-zero value.
- The index set D: To record which decision variables have been branched down, where branch down means that the variable takes a zero value.
- The indicator *next*: To indicate which variable will be processed in the child node (Note that such an indicator is used for producing the right child).
- The indicator *tindex*: To indicate the position of the branching variable in the branch order table I_t.
- The indicator $flag$: To indicate whether this node produces the right child, left child, or no child. If $flag$ is 0, then node produces the left child. If it is 1, the node produces the right child node, and if it is 2, then the node does not produce any child. This marker is used for the purpose of backtracking.

In the following, the notation $[X.x]$ is used to denote a member x at node X.

Suppose that all $v(Q_{S-1}^{[j]})$ have been calculated and all of the indices have been ranked according to their importance. Choose first s indices as the initial branching up variables and store them in I_t. The algorithm can then be illustrated as follows.

Initial step: Construct the *root* node of the enumeration tree. Let $[root.flag] = 0$, $[root.tindex] = 0$, and all of the other elements be $\emptyset$. Then add this *root* to *tree*. Pointer *cp* points to *root*: $cp = root$. Let the incumbent optimal value $CV = +\infty$ and set the current solution $Cy = null$. Go to step 1.

Step 1: If $cp = root$ and $[cp.flag] = 2$, then CV is the optimal value and Cy is the optimal solution. Otherwise go to step 2.

Step 2: Check $[cp.flag]$ to see whether it produces a left child or not. If $[cp.flag] = 1$, then go to step 4, otherwise, produce a node *new* and mark $[cp.flag] = 1$. Then, let $[new.D] = [cp.D]$, $[new.tindex] = [cp.tindex] + 1$ and $[newnode.pard] = cp$. Choose $I_s(new.tindex)$ to branch up. Thus, $[newnode.U] = [cp.U] \cup I_s(new.tindex)$. Mark $[cp.next] = I_s(new.tindex)$. Check $[new.U]$. If the number of the indices in $[new.U]$ is equal to s, then go to step 3, otherwise, add node *new* to the *Tree*, let $cp = new$, and repeat this step.

Step 3: Calculate the optimal value V and optimal solution y based on index set $[new.U]$. If $V < CV$, replace CV by V and cy by y. Go to step 4.

Step 4: Check $[cp.flag]$ to see whether or not it produces a right child. If $[cp.flag] = 2$, then go to step 5, otherwise, let $[cp.flag] = 2$ and produce a new node as *new*. Copy $[cp.U]$ to $[new.U]$ and let $[new.padr] = cp$, $[new.tindex] = [cp.tindex]$. Then, choose index $[cp.next]$ to branch down. Let $[new.D] = [cp.z] \cup [cp.next]$. Denote $\overline{[new.D]}$ as the complementary set of $[new.D]$. Calculate the optimal value BV and the optimal solution by based on index set $\overline{[new.D]}$. If $[BV > CV]$, then go to step 5, otherwise, check the number of indices in $[new.D]$. If it is equal to $S - s$, then let $CV = BV$, $y = by$, and go to step 5. If it is not, then update the branching order table I_t. Let $n = s - |new.U|$, which is the number of indices that must be updated in I_t. Suppose that

$$\{i_1, i_2, \ldots, i_n\} \subset \overline{new.D} \setminus new.U,$$

which satisfies

$$|By_{i_1}| \geq |By_{i_2}| \geq \ldots \geq |By_{i_n}| \geq Others.$$

I_t can then be updated as $I_t(new.tindex + 1) = i_1$, $I_t(new.tindex + 2) = i_2$, $\ldots, I_t(s) = i_n$. Add *new* to the *Tree* and let $cp = new$. Go to step 1.

Step 5: Let $cp = [cp.padr]$ and go to step 1.

In this algorithm, the rules for choosing an initial solution are based on the value of $v(Q_{S-1}^{[j]})$, and the variables with the largest value of $v(Q_{S-1}^{[j]})$ are branched first. The reason behind this selection rule is further explained in the next section. At each node, if a variable is chosen to branch, then this node produces two children. The left child is for branching up and the right child is for branching down. When dealing with left children, the information is updated

and stored, and the left children are produced one by one until s variables are branched up. When dealing with right children, based on Theorem 3.2, Corollary 3.3.1 and Corollary 3.3.2, the complementary set problem is calculated, which is a natural lower bound of this branch. The production of right children stops when $S - s$ variables are branched down, as branching down $S - s$ variables is equivalent to branching up s variables. If such a bound is better than the current incumbent, then the branching procedure continues and the branching order table is updated, otherwise, this branch is ended.

The solution process of this algorithm can be illustrated by considering the data that are given in Example 3.1. The process of implementing the algorithm is presented in Figure 3.1. Based on the value of $v(Q_s^{[j]})$, the initial branching table is $I_t = \{3, 5, 1, 2\}$.

The algorithm starts with the initial step. Construct the *root* and add the *root* to the *Tree*. The member *pard* at each node is not explicitly given and it can be assumed that such information has been stored.

node	*flag*	*next*	*tindex*	U	D
root	0	$\emptyset$	0	$\emptyset$	$\emptyset$

The algorithm then goes to step 1 and $cp = root$. As $[cp.flag]$ is 0, then go to Step 2. New node a_1 is produced as the left child. Mark $[cp.flag]$ as 1 and let $[a1.tindex] = [cp.tindex] + 1 = 1$. Thus, choose index $I_t(1) = 3$ to branch up. It follows that $[a_1.U] = \{3\}$. Mark $[cp.next]$ as 3. Add a_1 to the *Tree* and let $cp = a_1$. As Step 2 is repeated, the nodes a_2 and a_3 are subsequently added (see

Figure 3.1). The data of the *Tree* can be given as follows after a_3 is added.

node	*flag*	*next*	*tindex*	U	D
root	1	3	0	$\emptyset$	$\emptyset$
a_1	1	5	1	3	$\emptyset$
a_2	1	1	2	3, 5	$\emptyset$
a_3	0	2	3	1, 3, 5	$\emptyset$

$cp = a_3$ at Step 2. Produce a_4 with the data $[a_4.U] = \{1, 2, 3, 5\}$. The number of elements in $[a_4.U]$ is 4. Thus, go to Step 3 and carry out the calculation with the index set $I_4^{[j_1]}$=$\{1, 2, 3, 5\}$. The solution is $\{y_1, y_2, y_3, y_5\} = \{-15.1, -9.0, -12.8, -16.4\}$ with the objective value $v(Q_4^{[j_1]}) = -4766.14$. As $v(Q_4^{[j_1]})$ is less than CV, let $CV = -4766.14$ and $cy = y$.

The algorithm then goes to Step 4. Note that $cp = a_3$ and $[a_3.flag] = 1$. Produce a_5 as the right child. Let $[a_5.U] = [cp.U]$ and $[cp.flag] = 2$. As $[cp.next] = 2$, then $[cp.D] = \emptyset \cup \{2\} = \{2\}$. Now calculate the complementary problem based on $\overline{[cp.D]} = I_6^{[2]}$, which was already obtained in the initial step. It turns out that $BV = v(Q_6^{[2]}) = -5914.7 < CV$ and

$$\{by_1, by_3, by_4, by_5, by_6, by_7\} = \{-15.1, -12.4, -9.3, -16.2, -6.0, -8.1\}.$$

Branching order table I_t must now be updated. As $n = 4 - 3 = 1$, there is only one index that needs updating. Choose the index 4 that has the largest absolute solution value in $\overline{[a_5.D]} \setminus [a_5.U] = \{4, 6, 7\}$. Update the branching table as $I_t = \{3, 5, 1, 4\}$. As the number of element in $[a5.D]$ is 1, simply add this node to the *Tree*. Let $cp = a_5$ and go to Step 1. The data of the *Tree* are given now

as follows.

node	*flag*	*next*	*tindex*	U	D
root	1	3	0	$\emptyset$	$\emptyset$
a_1	1	5	1	3	$\emptyset$
a_2	1	1	2	3, 5	$\emptyset$
a_3	2	2	3	1, 3, 5	$\emptyset$
a_5	0	$\emptyset$	3	1, 3, 5	2

The algorithm arrives at Step 2. Produce new node a_6 with $[a_6.U] = \{1,3,5\} \cup I_t(4) = \{1,3,4,5\}$. Solving the optimization problem with index set $I_4^{[j_2]} = \{1,3,4,5\}$ yields $V = -4803.2 < CV$. Let $CV = -4803.2$ and $\{cy_1, cy_3, cy_4, cy_5\} = \{-15.1, -11.5, -9.6, -15.6\}$. Go to Step 4. Produce new node a_7. Set $[a_7.U] = \{1,3,5\}$, $[cp.next] = 4$, $[a_7.tindex] = [cp.tindex]$, $[a_7.D] = [cp.D] \cup [cp.next] = \{2,4\}$, and carry out the optimization problem with index set $\overline{[a_7.D]} = \{1,3,5,6,7\}$. It turns out that $BV = v(Q_5^{[j_1]}) = -5249.4 < CV$ and

$$\{by_1, by_3, by_5, by_6, by_7\} = \{-15.1, -12.9, -17.4, -5.2, -9.0\}.$$

The index that has the largest absolute value of by in index set $\overline{[a_7.D]} \setminus [a_7.U] = \{6,7\}$ is $\{7\}$, and thus the branching order table is updated as $I_t = \{1,3,5,7\}$. Add a_7 to the *Tree*. Let $cp = a_7$ and go to Step 1. The data of the enumeration tree is now as follows.

node	*flag*	*next*	*tindex*	U	D
root	1	3	0	$\emptyset$	$\emptyset$
a_1	1	5	1	3	$\emptyset$
a_2	1	1	2	3, 5	$\emptyset$
a_3	2	2	3	1, 3, 5	$\emptyset$
a_5	2	4	3	1, 3, 5	2
a_7	0	$\emptyset$	3	1, 3, 5	2, 4

As before, the algorithm repeats Step 2, Step 3, and Step 4 to produce the nodes a_8 and a_9. The index set $I_4^{[j_3]} = \{1, 3, 5, 7\}$ in node a_8 gives a better incumbent $CV = -5040.7$ with $\{cy_1, cy_3, cy_5, cy_7\}$=$\{-15.1, -11.5, -17.4, -10.4\}$. After exploring the node a_9, $[a_7.flag]$ becomes 2. Notice that $[a_5.flag] = 2$ and $[a_5.flag = 2]$. The algorithm backtracks to node a_2. At this stage, the data of $Tree$ are as follows.

node	*flag*	*next*	*tindex*	U	D
root	1	3	0	$\emptyset$	$\emptyset$
a_1	1	5	1	3	$\emptyset$
a_2	1	1	2	3, 5	$\emptyset$

As $[a_2.flag] = 1$, it is able to produce a right child a_{10} with $[a_{10}.D] = \{1\}$. Thus, $BV = v(Q_6^{[1]}) = -5236.1$ and

$$\{by_2, by_3, by_4, by_5, by_6, by_7\} = \{-8.7, -13.4, -9.1, -15.8, -5.8, -8.2\}.$$

The number of indices that must be updated in I_t is $n = 4 - 2 = 2$. Check the index set $\overline{[a_{10}.D]} \setminus [a_{10}.U] = \{2, 4, 6, 7\}$. Thus, $I_t = \{3, 5, 4, 2\}$. Add a_{10} to the $Tree$.

node	*flag*	*next*	*tindex*	U	D
root	1	3	0	$\emptyset$	$\emptyset$
a_1	1	5	1	3	$\emptyset$
a_2	1	1	2	3, 5	$\emptyset$
a_{10}	0	$\emptyset$	2	3, 5	1

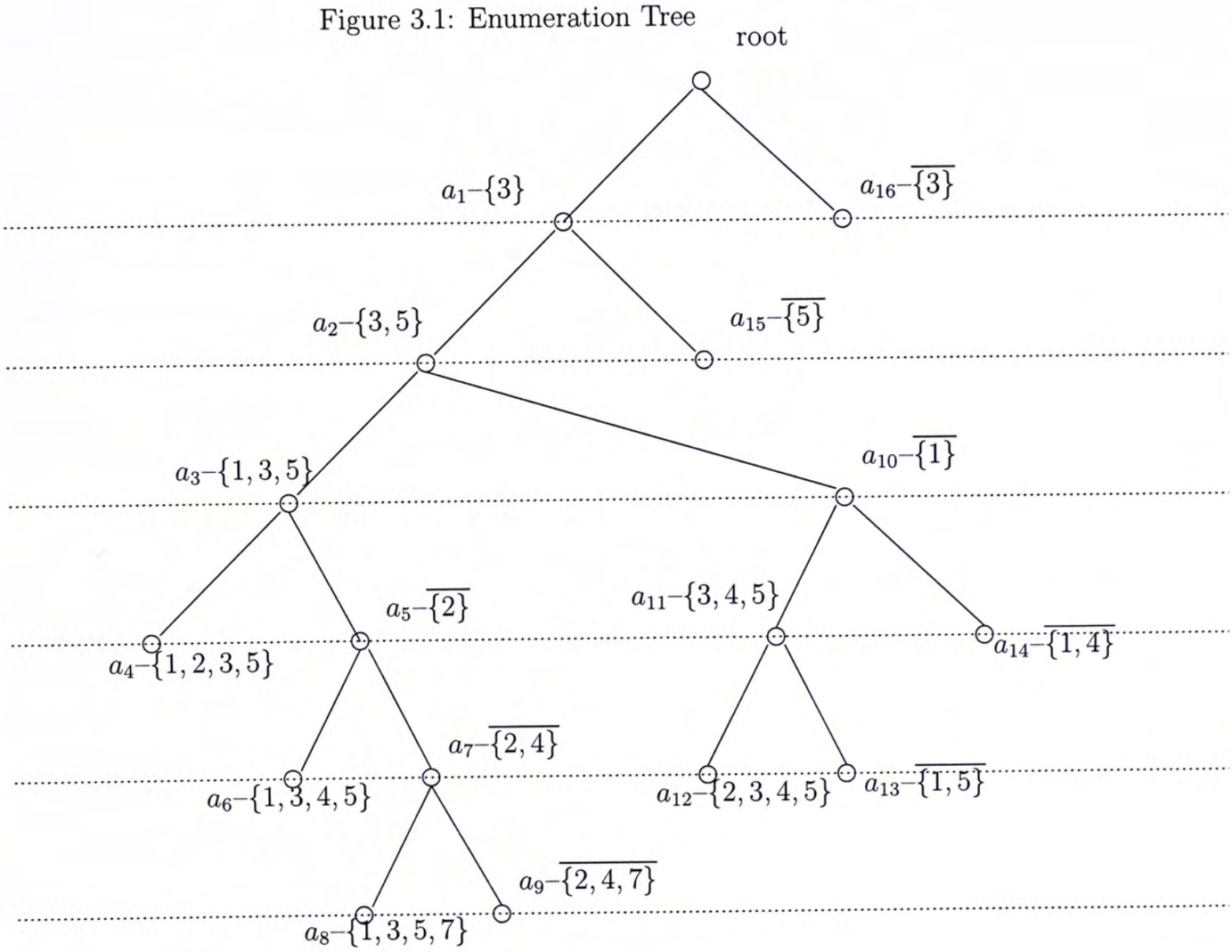
The algorithm continues to explore the children of a_{10}. The index set $[a_{12}.U] = I_4^{[j_5]} = \{2, 3, 4, 5\}$ gives $v(Q_4^{[j_4]}) = -4143.2 >$ CV, and thus this branch can be cut. For a_{13}, the index set $I_5^{[j_2]}$=$\overline{[a_{13}.D]}$=$\{2, 3, 4, 6, 7\}$ yields $v(Q_5^{[j_2]})$=$-4975.3 > CV$, and thus this branch can also be cut. For a_{14}, in-

dex set $I_5^{[j_3]}=\overline{[a_{14}.D]} = \{2,3,5,6,7\}$ and $v(Q_5^{[j_3]}) = -4589.5 > CV$, and thus this branch can be cut. At this stage, the data of $Tree$ are

node	*flag*	*next*	*tindex*	U	D
root	1	3	0	$\emptyset$	$\emptyset$
a_1	1	5	1	3	$\emptyset$

The algorithm goes back to check node a_1, the right child of which is a_{15} with $[a_{15}.D] = \{5\}$. It is known that $v(Q_6^{[5]}) = -4740.2 > CV$, and thus this branch can be cut. Finally, the algorithm returns to *root* and checks the right child a_{16} with $[a_{16}.D] = \{3\}$. Similarly, it is known that $v(Q_6^{[3]}) = -4530.7 > CV$, and thus this branch can be cut. The algorithm is terminated, and the optimal solution and optimal value are found to be cy and CV, respectively.

Figure 3.1: Enumeration Tree



3.3.2 Geometrical interpretation of the proposed ranking order

Computation experience shows that an initial solution or an initial ranking order plays an important role in any branch and bound algorithm. A good incumbent that is obtained at an early stage will facilitate the process of cutting out many branches. In this section, the idea behind the initial ranking order is explained.

Given a proper objective value, there is a contour in R^S space. Notice that

$$y'Gy + g'y = \frac{1}{2}(y + G^{-1}g)'G(y + G^{-1}g) - \frac{1}{2}g'G^{-1}g.$$

The objective contour of (Q_S) can be defined as follows for any $V \geq -\frac{1}{2}g'G^{-1}g$.

$$T(V) = \{y \mid V = y'Gy + g'y\}. \tag{3.3}$$

As $G \succ 0$, this contour is an ellipse in the R^S space with its origin at $-G^{-1}g$. The axes of this ellipse are not necessarily parallel to the x coordinates if G is not a diagonal matrix.

Let α and $\beta \in R^S$, such that

$$\alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_S \end{bmatrix}, \qquad \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_S \end{bmatrix}.$$

Denote the box (hyper-rectangle) by $[\alpha, \beta]$:

$$[\alpha, \beta] = \{x \mid \alpha_i \leq x_i \leq \beta_i \ , \ i = 1, 2.., S\}.$$

Lemma 3.3 *The minimum box that contains the ellipse* $T(V) = \{y \mid V = y'Gy + g'y\}$ *is* $[\alpha(V), \beta(V)]$, *where*

$$\begin{aligned} \alpha(V) &= -G^{-1}g - \sqrt{2V + g'G^{-1}g}Col\{\sqrt{d_{ii}}\} \\ \beta(V) &= -G^{-1}g + \sqrt{2V + g'G^{-1}g}Col\{\sqrt{d_{ii}}\}, \end{aligned}$$

where $D = (d_{ij})_{S\times S} = G^{-1}$ *and* $Col\{\sqrt{d_{ii}}\} = (\sqrt{d_{11}}, \sqrt{d_{22}}, \cdots, \sqrt{d_{SS}})$.

Proof. The *j-th* component of the upper corner $\beta(V)$ of the minimum box that contains $T(V)$ is the optimal value of the following optimization problem.

$$\begin{aligned} &\max && y_j \\ &Subject\ \ to: && V = \frac{1}{2}y'Gy + g'y. \end{aligned}$$

The Krush-Kuhn-Tucker conditions of this problem can be written as

$$e^j - \mu Gy - \mu g = 0 \tag{3.4}$$

$$\frac{1}{2}y'Gy + g'y = V, \tag{3.5}$$

where e^j is the *j-th* unit vector in R^S and $\mu \in R$ is a multiplier. Denote D_j as the *j-th* column of D for $j = 1, 2, \ldots, S$. Equation (3.4) implies that

$$y = \frac{1}{\mu}G^{-1}e^j - G^{-1}g = \frac{1}{\mu}D_j - G^{-1}g. \tag{3.6}$$

Substituting (3.6) into (3.5) yields

$$\mu = \sqrt{\frac{d_{jj}}{2V + g'G^{-1}g}}.$$

Thus, the optimal solution is

$$y = \sqrt{\frac{2V + g'G^{-1}g}{d_{jj}}}D_j - G^{-1}g$$

and the optimal value is

$$\beta_j(V) = \sqrt{d_{jj}[2V + g'G^{-1}g]} - D_j'g.$$

Similarly, solving

$$\begin{aligned} & \min & & y_j \\ & subject\ to: & & V = \frac{1}{2}y'Gy + g'y \end{aligned}$$

yields

$$\alpha_j(V) = -\sqrt{d_{jj}[2V + g'G^{-1}g]} - D_j'g$$

for $j = 1, 2, ..., S$.

When the value V decreases from positive infinity to $-\frac{1}{2}g'G^{-1}g$, both the contour $T(V)$ and the minimum box that contains $T(V)$ become smaller and eventually converge to a singleton. As we can see from Figure 3.2, when the box shrinks, the product $\alpha_j(V)\beta_j(V)$, $j = 0, 1, \ldots, S$, becomes positive one by one. It is easy to see that once $\alpha_j(V)\beta_j(V) > 0$, it will remain positive as V continues to decrease (see Figure 3.2). That is to say, variable y_j cannot take a zero value once $\alpha_j(V)\beta_j(V) > 0$. Let V_j^b denote the value V at which $\alpha_j(V)\beta_j(V) = 0$. The larger the value V_j^b, the earlier the occurrence of $\alpha_j(V)\beta_j(V) > 0$, and thus the earlier it is that y_i no longer takes a zero value. Therefore, the importance of each index can be measured by the value of V_j^b.

In Figure 3.2, the data are given as

$$Q = \begin{bmatrix} 2 & -1 \\ -1 & 3 \end{bmatrix}, \qquad g = \begin{bmatrix} -10 \\ -5 \end{bmatrix}.$$

Varying V from $V = 16$ to $V = -30$ makes the box shrink and the first index that does not change sign is x_1. Thus, the ranking order is $1, 2$.

Let $v(Q_{S-1}^{[j]})$ denote the optimal value of $(Q_{S-1}^{[j]})$ under the constraint of $y_j = 0$. As described in the previous section, the initial ranking order is based on the value $v(Q_{S-1}^{[j]})$. In the following, the relationship between V_j^b and $v(Q_{S-1}^{[j]})$ is investigated.

Theorem 3.3 *For problem (Q_s), $v(Q_{S-1}^{[j]}) = V_j^b$, where $j = 1, 2...S$.*

Proof. The objective contour of problem (Q_s) is defined in (3.3). Denote the center of the ellipse as $O = [O_1, O_2, \cdots, O_n]'$ and the inverse of G as $D = \{d_{i,j}\}$. It is clear that $O = -Db$. As proven in Lemma 3.3, the minimum box is $[\alpha(V), \beta(V)]$, where

$$\alpha_j(V) = O_j - \sqrt{(2V + g'G^{-1}g)d_{j,j}},$$
$$\beta_j(V) = O_j + \sqrt{(2V + g'G^{-1}g)d_{j,j}},$$

for $j = 1, 2, \ldots, S$. Notice that all $d_{jj} > 0$. Solving $\alpha_i(V)\beta_j(V) = 0$ for V yields,

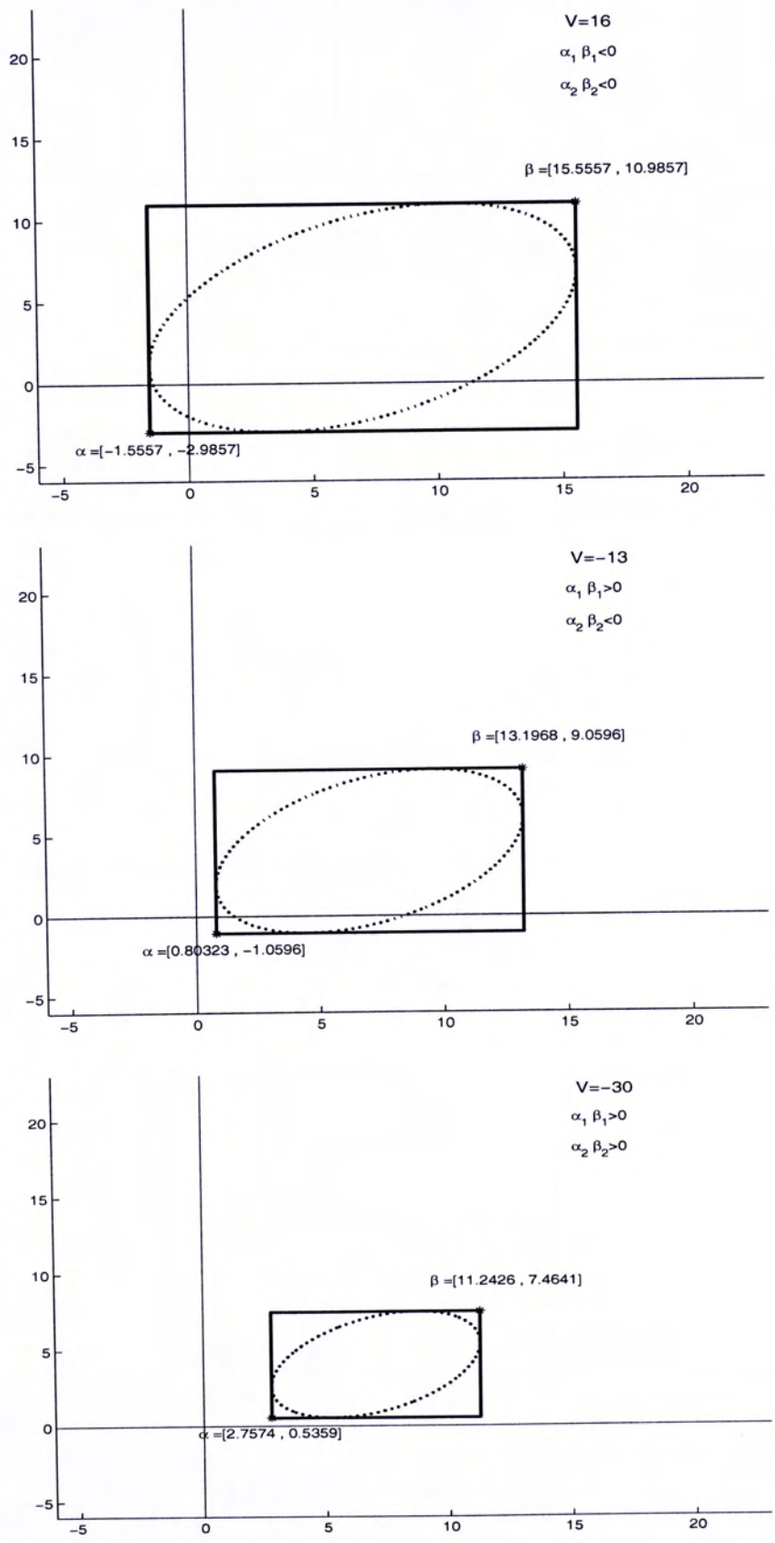
$$V_j^b = \frac{1}{2}(\frac{O_j^2}{d_{j,j}} - g'G^{-1}g). \tag{3.7}$$

For any j, the element of matrix G and vector g can be re-ordered and partitioned as

$$\overline{G} = \begin{bmatrix} G_{jj} & L' \\ L & H \end{bmatrix}, \quad \overline{g} = \begin{bmatrix} g_j \\ h \end{bmatrix},$$

where matrix H is formed by taking rows $1, 2, \ldots, j-1, j+1, \ldots, S$ and columns

Figure 3.2: Minimum Box



$1, 2, \ldots, j-1, j+1, \ldots, S$ of G, and

$$L = \begin{bmatrix} G_{1,j} \\ G_{2,j} \\ \vdots \\ G_{j-1,j} \\ G_{j+1,j} \\ \vdots \\ G_{S,j} \end{bmatrix}, \quad h = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_{j-1} \\ g_{j+1} \\ \vdots \\ g_S \end{bmatrix}.$$

As can be seen from (3.7), there are two parts in the expression of V_j^b. The part $g'G^{-1}g$ can be expressed as follows, as re-ordering does not change the value of $g'G^{-1}g$.

$$g'G^{-1}g = \overline{g}'\overline{G}^{-1}\overline{g} \tag{3.8}$$

$$= \begin{bmatrix} g_j & h \end{bmatrix} \begin{bmatrix} G_{j,j} & L' \\ L & H \end{bmatrix}^{-1} \begin{bmatrix} g_j \\ h \end{bmatrix}. \tag{3.9}$$

Notice the following well-known identity,

$$\begin{bmatrix} W_1 & W_2 \\ W_3 & W_4 \end{bmatrix}^{-1}$$

$$= \begin{bmatrix} (W_1 - W_2W_4^{-1}W_3)^{-1} & -(W_1 - W_2W_4^{-1}W_3)^{-1}W_2W_4^{-1} \\ -W_4^{-1}W_3(W_1 - W_2W_4^{-1}W_3)^{-1} & W_4^{-1} + W_4^{-1}W_3(W_1 - W_2W_4^{-1}W_3)^{-1}W_2W_4^{-1} \end{bmatrix}.$$

Thus, the inverse of $\overline{G}$ can be expressed as

$$\overline{G}^{-1} = \begin{bmatrix} \frac{1}{G_{j,j} - L'H^{-1}L} & -\frac{L'H^{-1}}{G_{j,j} - L'H^{-1}L} \\ -\frac{H^{-1}L}{G_{j,j} - L'H^{-1}L} & H^{-1} + \frac{H^{-1}LL'H^{-1}}{G_{j,j} - L'H^{-1}L} \end{bmatrix}. \tag{3.10}$$

Together with (3.8),

$$g'G^{-1}g = \frac{g_j^2 - 2g_jh'H^{-1}L + h'H^{-1}LL'H^{-1}h}{G_{j,j} - L'H^{-1}L} + h'H^{-1}h. \tag{3.11}$$

For the part $\frac{O_j^2}{d_{j,j}}$, $O = -Dg$ and (3.10) give the following,

$$O_j = \frac{g_j - L'H^{-1}h}{G_{j,j} - L'H^{-1}L}$$

and

$$d_{j,j} = \frac{1}{G_{j,j} - L'H^{-1}L}.$$

Thus,

$$\frac{O_j^2}{d_{j,j}} = \frac{(g_j - L'H^{-1}h)^2}{G_{j,j} - L'H^{-1}L}. \tag{3.12}$$

Using (3.11) and (3.12) in (3.7) yields

$$V_j^b = -\frac{1}{2}h'H^{-1}h.,$$

which is exactly the same as $v(Q_{S-1}^{[j]}) = -\frac{1}{2}h'H^{-1}h.$

The geometrical interpretation of the initial solution can be explained by using a box to approximate the objective contour. The larger the value of V_j^b, the higher possibility that this index will be included in the optimal index. This measure of the importance of the indices, of course, may not be totally accurate, but it can be used to determine the branching order of the indices at the initial stage along with some branch and bound rules to modify the ranking order until the exact solution is found.

Table 3.1 demonstrates the efficiency of the initial ranking order in a set of test problems. For each kind of test problem, 50 problems are randomly generated for testing. The procedure for generating such problems is similar to that which is mentioned in Section 3.4. The first two columns of Table 3.1 represent the (S, s) pair for the problem, and the third column is the average similarity

Table 3.1: Accuracy of the initial index set

S	s	*Similarity*	*Accuracy*
20	10	9	87.0%
30	15	13	85.0%
40	20	16	83.7%
50	25	20	82.3%
60	20	14	70.0%

between the initial solution and the optimal solution, which is round up to the nearest integer. The last column is the accuracy. The similarity and the accuracy are defined as follows. Let set I_t be the initial solution that is determined by $v(Q_{S-1}^{[j]})$ and the set I be the optimal index set. Then, $Similarity = |I_t \cap I|$ and $Accuracy = Similarity/|I|$, where $|I_t|$ and $|I|$ are the cardinality of I_t and I, respectively.

From Table 3.1, it can be seen that a high average accuracy can be achieved by using $v(Q_{S-1}^{[j]})$ as the importance measure. That is to say, most of the indices in the initial index set that is determined by $v(Q_{S-1}^{[j]})$ will be in the optimal index set. However, a significant amount of computation is needed to check the optimality of the solution.

3.3.3 Additional algorithmic ideas for enhancing computational efficiency

A major bottleneck in the computation process is caused by the calculation of the inverse matrix. To enhance the computational efficiency, a special method is adopted in this thesis. At each step of branching down, only one new variable is set to be zero, and thus the new inverse can be obtained by modifying the inverse at the parent node.

Lemma 3.4 *Take a nonsingular symmetric matrix $G_n \in R^{n\times n}$ and its inverse D_n. For a $j \in \{1, 2, \ldots, n\}$, construct two $(n-1)\times(n-1)$ dimensional matrices G_{n-1} and D_{n-1}, by deleting the j-th column and the j-th row of G_n and the j-th column and the j-th row of D_n, respectively. Let the column vector $d \in R^{(n-1)}$ be formed by the j-th column of D_n with dropping element $D_{j,j}$, and let ω denote $D_{j,j}$. The inverse of G_{n-1} is then $G_{n-1}^{-1} = D_{n-1} - \frac{1}{\omega}dd'$, which can be realized with no more than $3(n-1)$ elementary row operations.*

Proof. Without loss of generality, only the situation for $j = 1$ is proved. The partition of G_n and D_n is

$$G_n = \begin{bmatrix} \nu & g' \\ g & G_{n-1} \end{bmatrix}, \qquad D_n = \begin{bmatrix} \omega & d' \\ d & D_{n-1} \end{bmatrix},$$

where g is an $(n-1)$ dimensional column vector and ν is a scalar.

Note the following relationship,

$$D_n G_n = \begin{bmatrix} \nu\omega & \omega g' + d' G_{n-1} \\ \nu d + D_{n-1} g & g d' + D_{n-1} G_{n-1} \end{bmatrix}. \tag{3.13}$$

By the definition of the inverse matrix, it is known that

$$D_{n-1} G_{n-1} = I_{n-1} - g d'. \tag{3.14}$$

As g and d are vectors, gd' is a rank one matrix. Let g_i and d_i be the i-th component of g and d, respectively. Construct matrix E as

$$E = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ -\frac{g_2}{g_1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{g_{n-1}}{g_1} & 0 & \cdots & 1 \end{bmatrix}.$$

Then,

$$E(I - gd') = \begin{bmatrix} 1 - g_1 d_1 & g_1 d_2 & \cdots & g_1 d_{n-1} \\ -\frac{g_2}{g_1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{g_{n-1}}{g_1} & 0 & \cdots & 1 \end{bmatrix}.$$

Thus, only the elementary row operation need to be used to eliminate first the row and the first column, which requires $2(n-1)$ row operations. Together with the elementary row operation for E, there are in total $3(n-1)$ elementary row operations, which is linear to the size of problem.

In equation (3.13), it is known that

$$\omega g' + d' G_{n-1} = 0_{1\times(n-1)},$$

which becomes $\omega dg' + dd' G_{n-1} = 0_{1\times(n-1)}$ after pre-multiplying d on both sides. Substituting it into (3.14) yields

$$(D_{n-1} - \frac{1}{\omega} dd') G_{n-1} = I_{n-1},$$

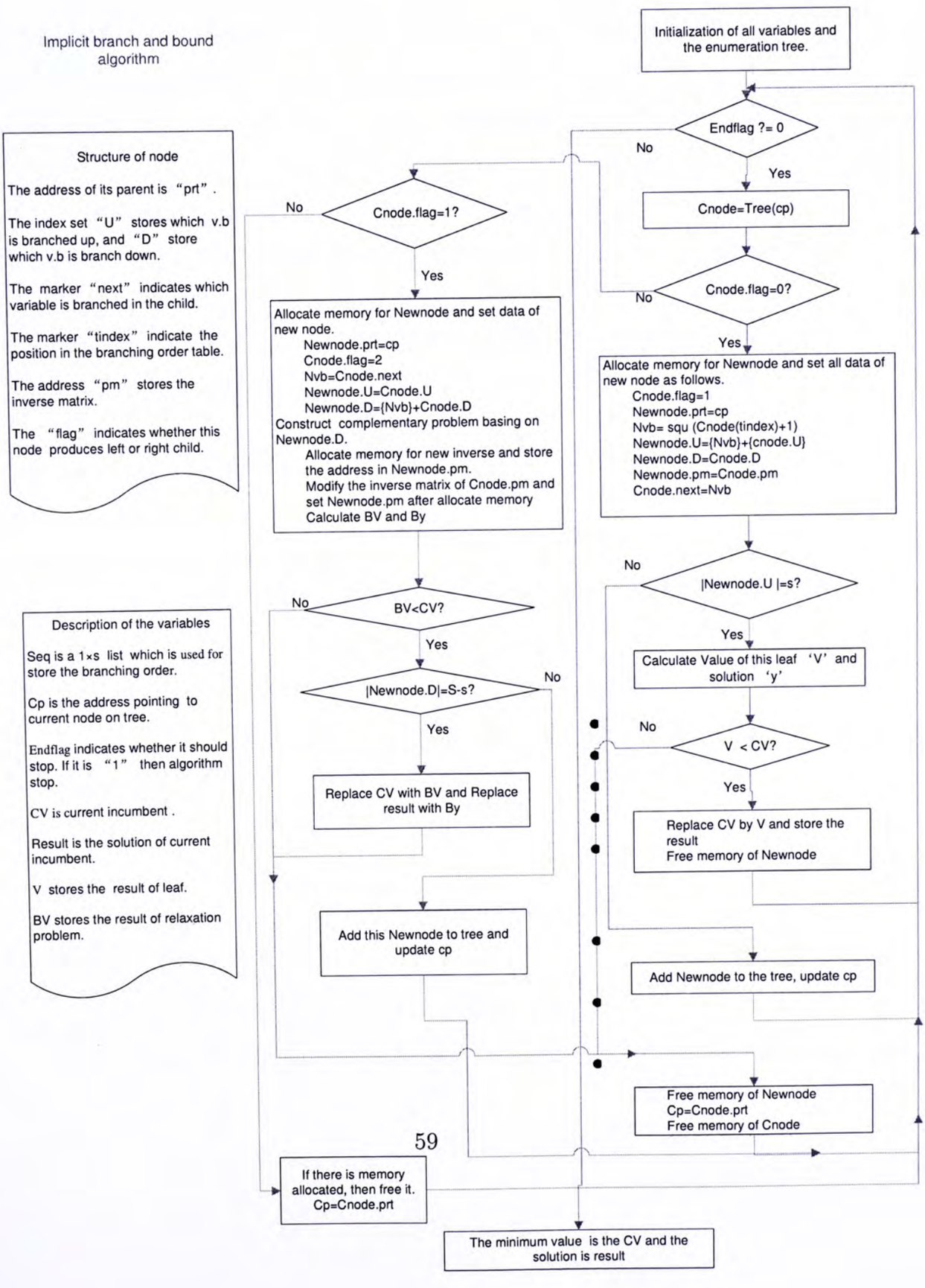
which explicitly gives the inverse of G_{n-1}.

Actually, this lemma provides two methods for calculating G_{n-1}^{-1}. The first uses elementary row operations to obtain matrix $I - gd'$, which requires the storage of the vector d at the parent node to implement the algorithm. The second

method gives the explicit expression of G_{n-1}^{-1}, but requires the storage of every D matrix in the parent node, which takes up more memory. However, both of these two methods are more efficient than the approach of calculating G_{n-1}^{-1} directly.

The second method is used in the computation in this thesis. The computation experiment shows that applying such method to the children problem of each node significantly reduces the computational burden in calculating the inverse. Incorporating this computational method into the algorithm, the flow chart of the improved algorithm is given in Figure 3.3.

Figure 3.3: Flow Chart of the Proposed Algorithm



3.4 Numerical example and computational results

In this section, an example is presented to show how to solve problem (P) using the proposed method. The computational performance of this method and CPLEX 9.1, which is a commercial optimization software, is then compared.

The solution algorithm that was developed in the previous section for cardinality constrained convex quadratic programming problems can be readily applied to problem (P). Consider the following illustrative example for a discrete-time linear-quadratic control problem with set-up costs that are attached to nonzero control actions

$$\begin{aligned}
&\min 2500 \sum_{t=0}^{6} Supp(u(t)) + x'(7)Q(7)x(7) \\
&+ \sum_{t=0}^{6} [x'(t)Q(t)x(t) + u'(t)R(t)u(t)] \\
&\text{Subject to : } x(t+1) = A(t)x(t) + B(t)u(t) \\
&x(0) = (-2.16, -8.33)',
\end{aligned}$$

where

$$A(0) = \begin{pmatrix} 0.125 & -1.147 \\ 0.288 & 1.191 \end{pmatrix}, A(1) = \begin{pmatrix} -1.147 & 1.190 \\ 1.191 & -0.038 \end{pmatrix}, A(2) = \begin{pmatrix} -1.147 & 1.189 \\ 1.191 & -0.038 \end{pmatrix},$$

$$A(3) = \begin{pmatrix} -1.147 & 1.189 \\ 1.191 & -0.038 \end{pmatrix}, A(4) = \begin{pmatrix} -1.147 & 1.189 \\ 1.191 & -0.038 \end{pmatrix}, A(5) = \begin{pmatrix} -1.147 & 1.189 \\ 1.191 & -0.038 \end{pmatrix},$$

$$A(6) = \begin{pmatrix} -1.147 & 1.189 \\ 1.191 & -0.038 \end{pmatrix};$$

$$B(0) = \begin{pmatrix} -0.612 & 1.054 \\ -2.355 & 2.419 \end{pmatrix}, B(1) = \begin{pmatrix} -15.705 & 0.661 \\ -60.473 & 1.5167 \end{pmatrix}, B(2) = \begin{pmatrix} -13.666 & 2.386 \\ -52.619 & 5.476 \end{pmatrix},$$

$$B(3) = \begin{pmatrix} -8.670 & 0.576 \\ -33.384 & 1.322 \end{pmatrix}, B(4) = \begin{pmatrix} -20.689 & 2.407 \\ -79.664 & 5.526 \end{pmatrix}, B(5) = \begin{pmatrix} -12.826 & 5.548 \\ -49.386 & 12.734 \end{pmatrix},$$

$$B(6) = \begin{pmatrix} -18.161 & 2.712 \\ -69.928 & 6.225 \end{pmatrix};$$

$$Q(0) = \begin{pmatrix} 29.31 & 4.01 \\ 4.01 & 44.80 \end{pmatrix}, Q(1) = \begin{pmatrix} 13.17 & 0.58 \\ 0.58 & 17.39 \end{pmatrix}, Q(2) = \begin{pmatrix} 7.55 & -4.42 \\ -4.42 & 19.90 \end{pmatrix},$$

$$Q(3) = \begin{pmatrix} 15.06 & -3.77 \\ -3.77 & 4.73 \end{pmatrix}, Q(4) = \begin{pmatrix} 20.45 & 0.25 \\ 0.25 & 21.96 \end{pmatrix}, Q(5) = \begin{pmatrix} 22.97 & 2.89 \\ 2.89 & 36.84 \end{pmatrix},$$

$$Q(6) = \begin{pmatrix} 42.00 & -1.91 \\ -1.91 & 31.93 \end{pmatrix}, Q(7) = \begin{pmatrix} 23.66 & -0.09 \\ -0.09 & 23.26 \end{pmatrix}; R(0) = \begin{pmatrix} 1747.44 & -316.82 \\ -316.82 & 762.43 \end{pmatrix},$$

$$R(1) = \begin{pmatrix} 5.65 & 0.31 \\ 0.31 & 28.17 \end{pmatrix}, R(2) = \begin{pmatrix} 36.91 & 3.53 \\ 3.53 & 39.16 \end{pmatrix}, R(3) = \begin{pmatrix} 1511.68 & -86.56 \\ -86.56 & 476.25 \end{pmatrix},$$

$$R(4) = \begin{pmatrix} 24.58 & 1.74 \\ 1.74 & 36.16 \end{pmatrix}, R(5) = \begin{pmatrix} 24.30 & -0.76 \\ -0.76 & 0.43 \end{pmatrix}, R(6) = \begin{pmatrix} 938.85 & -0.43 \\ -0.43 & 913.22 \end{pmatrix}.$$

The control problem is first converted into a cardinality constrained convex quadratic programming problem. The data of G and g are given in Figure 3.4. Note that in this example the control vector is two dimensional. Although the resulting y vector actually 14 dimensions, the two variables that are associated with each $u(t)$ are bundled together and are dealt with as a single entity. Thus, in the cardinality constrained problem, s varies from 1 to $S = 7$. Solving problem (Q_s) for $1 \leq s \leq 7$ yields the optimal index set for each s and the corresponding value of $v(Q_s)$, which are shown in Table 3.2 where the set-up cost, ws, is also calculated for different s. In the calculation of $v(Q_s)$, the algorithm that was developed in the previous section is used. The solution process only requires 15 unconstrained quadratic optimizations, whereas the total enumeration requires

the evaluation of

$$\sum_{s=1}^{7} C_7^{(s)} = 2^7 - 1 = 127$$

unconstrained convex quadratic optimization problems.

A comparing of the values of the sum of $v(Q_s) + ws$ identifies the optimal cardinality for the example problem

$$s^* = 3$$

and the corresponding optimal control

$$u_0 = 0,\ u_1 = \begin{pmatrix} 0.67 \\ 6.26 \end{pmatrix},$$

$$u_2 = \begin{pmatrix} -1.91 \\ -8.41 \end{pmatrix},\ u_3 = \begin{pmatrix} 0.58 \\ 0.08 \end{pmatrix},$$

$$u_4 = u_5 = u_6 = 0.$$

To evaluate the computational performance of the proposed solution scheme, this method is compared with the CPLEX 9.1 solver for the solution of problem (Q_s). All of the problems are randomly generated using the following procedure. For matrix G, generate a diagonal matrix Λ, the elements of which are uniformly distributed in a range of $(0, 50]$, then generate orthogonal matrix Γ, the elements of which are normally distributed with a 0 mean and a standard deviation of 50. This orthogonal matrix is generated by the Gram-Schmidt procedure. Finally, the matrix G is given by $\Gamma'\Lambda\Gamma$. The elements of vector b are uniformly distributed in a range of $[-400, 400]$.

The results for five different kinds of problems are listed in Table 3.3. In this table, the first column shows the "size" of the problem that corresponds to the

Figure 3.4: Data of G and g

$$
G = \begin{bmatrix}
99514.0 & -53479.1 & -12805539.6 & 17019.7 & 596470.0 & -33587.4 & -188108.3 & 3807 & 263263.4 & -10565.9 & -66321.5 & 7895.0 & 38011.3 & -1824.4 \\
-53479.1 & 30721.8 & 698823.0 & -9241.4 & -329428.8 & 18680.9 & 101485.1 & -2014.1 & -147763.5 & 6049.5 & 35225.3 & -4006.9 & -21427.0 & 1058.2 \\
-12805539.6 & 698822.1 & 17498306.7 & -236534.6 & -7817682.6 & 429122.4 & 2669869.2 & -574119.8 & -3249707.8 & 120305.7 & 988493.6 & -133504.9 & -461385.9 & 19613.8 \\
17019.7 & -9241.4 & -236534.6 & 3293.8 & 102312.9 & -5499.2 & -37094.5 & 830.5 & 40415.4 & -1383.0 & -14192.7 & 2063.50 & 5650.6 & -211.4 \\
596470.0 & -329428.8 & -7817682.6 & 102312.9 & 3838809.8 & -224477.9 & -1057238.4 & 18861.4 & 1844951.0 & -81639.2 & -337353.9 & 28278.5 & 272253.8 & -14966.6 \\
-33587.4 & 18680.9 & 429122.4 & -5499.2 & -224477.9 & 13750.4 & 51795.4 & -734.2 & -117742.9 & 5666.2 & 13876.6 & -180.2 & -17727.5 & 1086.1 \\
-188108.3 & 101485.1 & 2669869.2 & -37094.5 & -1057238.4 & 51795.4 & 498003.6 & -12936.0 & -297131.7 & 3702.4 & 212849.0 & -38203.7 & -36544.2 & -302.8 \\
3807.0 & -2014.1 & -57411.8 & 830.5 & 18861.4 & -734.2 & -12936.0 & 1326.8 & 1143.4 & 301.6 & -6119.8 & 1272.2 & -103.6 & 91.9 \\
263263.4 & -147763.5 & -3249707.8 & 40415.4 & 1844951.0 & -117742.9 & -297131.7 & 1143.4 & 1422837.3 & -82392.5 & 66039.42 & -47436.4 & 224983.9 & -17125.2 \\
-10565.9 & 6049.5 & 120305.7 & -1383.0 & -81639.2 & 5666.2 & 3702.4 & 301.6 & -82392.5 & 5323.2 & -11866.0 & 4454.1 & -13399.6 & 1129.4 \\
-66321.5 & 35225.3 & 988493.6 & -14192.7 & -337353.9 & 13876.6 & 212849.0 & -6119.8 & 66039.4 & -11866.0 & 264666.8 & -66519.7 & 80904.7 & -9481.3 \\
7895.0 & -4006.9 & -133504.9 & 2063.5 & 28278.5 & -180.2 & -38203.7 & 1272.2 & -47436.4 & 4454.1 & -66519.7 & 17778.4 & -27354.2 & 2889.0 \\
38011.3 & -21427.0 & -461385.9 & 5650.6 & 272253.8 & -17727.5 & -36544.2 & -103.6 & 224983.9 & -13399.6 & 80904.7 & -27354.2 & 244489.4 & -22525.3 \\
-1824.4 & 1058.2 & 19613.8 & -211.4 & -14966.6 & 1086.1 & -302.8 & 91.9 & -17125.2 & 1129.4 & -9481.3 & 2889.0 & -22525.3 & 3970.9
\end{bmatrix}
$$

$$
g = \begin{bmatrix} 1721764.0 & -9444119.0 & -23129650.8 & 308734.3 & 10663047.4 & -596753.1 & -3430728.4 & 70554.3 & 4639616.8 & -182845.5 & -1225252.9 & 151117.5 & 667291.6 & -31186.7 \end{bmatrix}'
$$

Table 3.2: Solutions to (Q_s)

s	Optimal index set	$v(Q_s)$	ws
1	$\{3\}$	101,013.899	2,500
2	$\{2,3\}$	28,815.6693	5,000
3	$\{2,3,4\}$	23,651.8494	7,500
4	$\{1,2,3,4\}$	21,493.9760	10,000
5	$\{1,2,3,4,5\}$	21,402.2807	12,500
6	$\{1,2,3,4,5,6\}$	21,332.0927	15,000
7	$\{1,2,3,4,5,6,7\}$	21,332.0909	17,500

dimension of G, the second column shows the cardinality size, and the third column is the combinatorial number C_S^s if all cases are enumerated. The notation "$S-s$" is used to denote a case with a problem size of S and a cardinality size of s. It is not possible to solve any of these problems, except for "30-15", using the enumeration method in a given time length of 3600 CPU seconds, even with a fast machine such as SUN blade workstation (2 Gmhz).

Table 3.3: Problem types

S	s	Total Number
30	15	1.1512×10^{8}
40	20	1.3785×10^{11}
50	25	4.7129×10^{13}
60	20	4.1918×10^{15}
100	20	5.3598×10^{20}

For the CPLEX 9.1 implementation, the problem (Q_s) is rewritten in the following formula.

$$
\begin{aligned}
(cplex - Q_s) \qquad v(s) = \min \quad & \frac{1}{2} y'Gy + g'y \\
s.t \quad & y_i + Mz_i \geq 0 \qquad i = 1, 2...S \\
& y_i - Mz_i \leq 0 \qquad i = 1, 2...S \\
& \sum_{i}^{S} z_i \leq s \\
& z_i \in \{0, 1\} \qquad i = 1, 2...S,
\end{aligned}
$$

where M is a large positive number. Both z_i for $i = 1, 2, \ldots, S$ and M are introduced to convert (Q_s) into a mixed integer programming formulation. After several tries, it was found that $M = 1000$ is large enough to cover the optimal solutions for the computation. CPLEX 9.1 is designed to deal with a general class of mixed-integer problems, such as linear and quadratic mixed-integer problems. The CPLEX user manual mentions that the program uses disjunctive cuts and cover cuts.

For each kind of problem that is listed in Table 3.3, 20 cases are generated for calculation. In Tables 3.4, 3.5, 3.6, 3.7, and 3.8, the column "CPU" refers to the CPU execution time in seconds. An upper limit of 3600 CPU seconds is

pre-set for calculation, which means that the calculation is stopped if a solution is not found within 3600 CPU seconds. The column "node" corresponds to the number of nodes that is visited in the enumeration tree. The column "speed" is calculated by "node"/"CPU", which measures the transition speed from one node to another. For the proposed method, the accurate rate of the initial index set in column "Acc" is given, as detailed in Section 3.3.2.

Tables 3.4 and 3.5 show that both the proposed method and the CPLEX 9.1 obtain exact solutions for problems "30-15" and "40-20" within the 3600-second time limit. The columns "CPU" and "node" show that the proposed method uses less time and checks less nodes than CPLEX 9.1, and has a higher transition "Speed". It is thus apparent that the proposed method is more efficient.

Furthermore, Tables 3.6 and 3.7 show that most of the problems of sizes "50-25" and "60-20" can be solved by the proposed method within the given time limit, whereas CPLEX 9.1 is unable to solve most of them with this limit, especially for the case "60-20" in Table 3.7. Although CPLEX 9.1 finds the optimal solutions for some of the problems, it is unable to prove the optimality within the given time limit, and only solves two problems exactly.

For problems of a large "size", such as "100-20" in Table 3.8, neither the proposed method nor CPLEX 9.1 are able to find exact solutions within the time limit. However, it can be seen that the incumbent of the proposed method is better than or equal to CPLEX 9.1 in 17 cases out of 20, and the proposed method has a higher transition speed.

The main bottleneck that occurs both with the proposed method and

CPLEX 9.1 is caused by the number of nodes that must be visited to prove the optimality. Even when the optimal solution can be found at the beginning, the branch and bound tree can still grow to several million nodes before termination, even for a moderate-sized problem such as "40-20". However, the proposed bounding strategy provides a relatively efficient pruning process for the enumeration tree, which is why the proposed method visits less nodes than CPLEX 9.1 to reach optimality. Furthermore, the proposed method of calculating the inverse matrix for relaxation problem enhances the transition speed.

Table 3.4: Experiment for "30-15"

	IMPLICIT Branch and Bound					CPLEX			
Prob	CPU	Node	Speed	Acc	OptV	CPU	Node	Speed	OptV
1	0.04	195	4875	0.93	−2566	0.20	266	1538	−2566
2	0.02	100	5000	0.93	−1769	0.11	169	1536	−1769
3	0.83	4120	4964	0.73	−1409	4.48	6862	1530	−1409
4	0.26	1257	4903	0.80	−2414	1.24	1797	1449	−2414
5	0.58	2846	4907	0.80	−2215	2.89	4398	1476	−2215
6	0.15	698	4653	0.93	−3138	1.11	1564	1409	−3138
7	0.08	406	5075	0.87	−2146	0.57	714	1253	−2146
8	0.64	3103	4848	0.80	−1670	3.67	5825	1587	−1670
9	0.20	961	4805	0.87	−3177	1.39	2089	1503	−3177
10	0.09	416	4622	1.00	−1231	0.52	735	1414	−1231
11	0.06	296	4933	0.87	−2899	0.40	5161	1290	−2899
12	0.20	983	4915	0.87	−2010	1.65	2438	1478	−2010
13	0.12	589	4908	0.87	−2111	0.96	1402	1460	−2111
14	0.46	2288	4974	0.60	−3202	3.17	4631	1461	−3202
15	0.09	445	4944	0.87	−2938	0.52	687	1321	−2938
16	0.32	1551	4847	0.67	−1964	1.09	1613	1480	−1964
17	0.12	575	4792	0.87	−2135	0.69	911	1320	−2135
18	0.57	2829	4963	0.80	−3122	3.35	4985	1488	−3122
19	0.03	126	4200	0.93	−2398	0.31	310	1000	−2398
20	0.03	128	4267	1.00	−5404	0.26	303	1165	−5404
Avg	0.245	1195	4878	0.85	–	1.44	2111	1466	–

Table 3.5: Experiment for "40-20"

	IMPLICIT Branch and Bound					CPLEX			
Prob	CPU	Node	Speed	Acc	OptV	CPU	Node	Speed	OptV
1	1.97	4906	2490	0.75	−3599	8.72	9418	1080	−3599
2	2.33	5781	2481	0.85	−3144	9.58	9700	1013	−3144
3	2.66	6663	2505	0.90	−4209	8.99	9248	1029	−4209
4	3.39	8459	2495	0.75	−3267	11.34	11994	1058	−3267
5	2.04	5120	2510	0.85	−2231	9.94	10192	1025	−2232
6	0.40	1036	2590	0.95	−7878	2.42	6059	2504	−7878
7	3.60	9041	2511	0.85	−4415	16.39	16894	1031	−4415
8	2.10	5219	2485	0.95	−3092	15.67	17120	1093	−3092
9	0.25	633	2532	0.90	−3777	0.89	815	916	−3777
10	3.50	8834	2524	0.90	−1791	15.07	16208	1076	−1791
11	4.49	11287	2514	0.90	−3019	21.93	24157	1102	−3018
12	0.24	591	2462	0.90	−4373	1.03	814	790	−4373
13	0.73	1788	2449	0.85	−5469	2.67	2604	975	−5468
14	1.14	2792	2449	0.85	−3905	4.00	4100	1025	−3905
15	1.00	2469	2469	0.90	−2210	5.26	5442	1035	−2210
16	0.48	1202	2504	0.80	−5192	5.28	5201	985	−5292
17	3.22	8037	2496	0.75	−5342	17.51	18433	1053	−5342
18	3.04	7630	2510	0.90	−7376	21.17	22746	1074	−7376
19	4.69	11893	2536	0.75	−3343	10.43	11156	1070	−3343
20	4.45	11203	2518	0.90	−2465	16.77	18341	1094	−2465
Avg	2.29	5729	2506	0.857	–	10.25	11032	1076	–

Table 3.6: Experiment for "50-25"

	IMPLICIT Branch and Bound					CPLEX			
Prob	CPU	Node	Speed	Acc	OptV	CPU	Node	Speed	OptV
1	93.8	139936	1492	0.84	−3764	472.3	377994	800	−3763
2	134.5	201847	1501	0.68	−4686	533.5	429403	805	−4686
3	9.0	13286	1475	0.84	−8446	39.9	29417	737	−8444
4	7.7	11267	1456	0.92	−5460	19.9	14937	752	−5461
5	67.3	100731	1497	0.88	−10123	257.2	199983	778	−10123
6	35.4	52017	1468	0.76	−4599	147.0	111496	759	−4599
7	7.3	10875	1496	0.84	−7438	56.4	42183	748	−7438
8	9.0	13286	1476	0.84	−8446	40.1	29417	733	−8446
9	252.1	381174	1512	0.80	−8004	1240.1	973654	785	−8004
10	15.1	21940	1455	0.88	−15737	76.1	55400	728	−15736
11	9.7	13550	1448	0.84	−5359	50.2	39729	792	−5359
12	13.9	20396	1463	0.96	−7084	53.1	40333	760	−7083
13	80.9	121703	1505	0.88	−7481	312.1	235800	756	−7480
14	148.1	223882	1511	0.76	−4624	417.1	318545	764	−6424
15	7.2	10333	1443	0.96	−6411	36.3	28124	775	−6411
16	8.9	13299	1493	0.80	−12950	38.7	30229	781	−12948
17	3.3	4755	1445	0.88	−13449	14.0	10533	751	−13448
18	15.5	22612	1458	0.80	−13622	60.8	47162	776	−13622
19	39.8	58925	1482	0.92	−3858	350.1	271508	775	−3858
20	18.4	27093	1471	0.76	−8239	60.4	46368	767	−8239
Avg	48.829	73145	1501	0.842	–	213.8	166610	779.4	–

Table 3.7: Experiment for "60-20"

	IMPLICIT Branch and Bound					CPLEX			
Prob	CPU	Node	Speed	Acc	OptV	CPU	Node	Speed	OptV
1	1789.1	2882292	1611	0.65	−5879	3600.0	1971850	548	−5845
2	976.9	1550888	1588	0.75	−10035	3437.0	1914699	557	−10034
3	3146.3	4971152	1580	0.80	−4780	3600.0	2328831	647	−4779
4	771.1	1240238	1608	0.80	−10974	3600.0	2038739	566	−10974
5	1584.7	2506706	1582	0.75	−6044	3600.0	2090008	581	−6045
6	3600.0	5791477	1609	0.75	−8966	3600.0	2097874	583	−8955
7	3228.2	5381900	1667	0.30	−6372	3600.0	2081549	578	−6055
8	1148.0	1801659	1569	0.75	−5194	3600.0	2261514	628	−5194
9	3600.0	6077933	1688	0.75	−6238	3600.0	2248350	625	−6237
10	3.4	4874	1442	0.95	−18712	19.4	9797	505	−18712
11	2594.3	4075817	1571	0.70	−5724	3600.0	2166104	602	−5725
12	377.7	606946	1607	0.80	−5974	3600.0	2468798	686	−5974
13	3491.7	5561049	1593	0.45	−4893	3600.0	2375712	660	−4893
14	1787.9	2810716	1572	0.80	−5035	3600.0	2189012	608	−5035
15	3600.0	6015737	1671	0.55	−5905	3600.0	2164438	601	−5902
16	1221.2	1927594	1578	0.70	−5379	3600.0	2189241	608	−5379
17	2354.6	3774418	1603	0.80	−5609	3600.0	2102171	559	−5609
18	67.9	99727	1468	0.70	−6433	792.7	443368	584	−6433
19	1057.8	1651446	1561	0.90	−6002	3600.0	2241343	623	−6000
20	996.3	1558087	1564	0.75	−5657	3600.0	2134921	593	−5657
avg	1869.9	3014532	1612	0.700	–	3272	1975900	604	–

Table 3.8: Experiment for "100-20"

	IMPLICIT Branch and Bound					CPLEX			
Prob	CPU	Node	Speed	Acc	OptV	CPU	Node	Speed	OptV
1	3600	3254805	904	0.75	−6855.0	3600	1125201	313	−6392
2	3600	3332764	926	0.80	−5251.0	3600	1066701	296	−4958
3	3600	3204344	890	0.75	−6099	3600	1018361	283	−6099
4	3600	3374212	937	0.70	−6028	3600	1009624	281	−6544
5	3600	2983456	829	0.85	−5315	3600	1153369	320	−4911
6	3600	3000456	833	0.90	−6583	3600	986664	274	−6583
7	3600	3413072	948	0.65	−5031	3600	878901	244	−5031
8	3600	3011222	836	0.75	−6722	3600	1361740	378	−5646
9	3600	2899322	805	0.55	−39333	3600	1027848	286	−28900
10	3600	3004543	835	0.60	−7033	3600	1099801	306	−7060
11	3600	2978562	827	0.50	−5119	3600	1126426	313	−5688
12	3600	3054333	848	0.65	−9037	3600	984410	273	−9037
13	3600	3012112	836	0.80	−6788	3600	1358008	377	−5102
14	3600	3056245	848	0.60	−5100	3600	1383312	384	−5101
15	3600	3232727	897	0.70	−4344	3600	1413923	393	−4344
16	3600	3198994	888	0.70	−5232	3600	1288321	358	−4412
17	3600	3124543	868	0.75	−7232	3600	1208283	336	−6679
18	3600	3124434	867	0.80	−6678	3600	1208283	336	−6679
19	3600	3023273	839	0.70	−8410	3600	806521	224	−8405
20	3600	3093305	859	0.90	−8543	3600	1229834	342	−6735
avg	3600	3293784	915	0.78	–	3600	1136800	316	–

Chapter 4

Summary and Future Work

In this thesis, the cardinality constrained discrete-time linear-quadratic control problem (P_s) is studied. Such a problem with limits on the number of times that control can be implemented is motivated by the need to solve discrete-time linear-quadratic problem with a set-up cost for control actions.

In Chapter 2, the feasibility of using dynamic programming as a solution scheme is investigated, and for scalar-state situations an elegant solution structure is revealed. In the derived analytical solution, a two-dimensional recursive equation is solved backward to achieve the optimal control policy. Dynamic programming fails to solve the general cardinality constrained discrete-time linear-quadratic control problem in multi-dimensional-state situations, because of the state dependency in solving the Riccatti equation.

In Chapter 3, the cardinality constrained discrete-time linear-quadratic control problem with a multi-dimensional state is reformulated as a cardinality constrained quadratic programming problem (Q_s). This kind of problem has been studied in the area of subset regression and portfolio selection. By studying the

structure of the cardinality constrained quadratic programming problem, an efficient branch and bound solution method is developed in this thesis, the success of which is largely due to the procedure for generating the initial solution index.

Despite the effectiveness of the method that is proposed in this thesis, many issues remain to be explored in future research. Cardinality constrained discrete-time linear-quadratic *stochastic* control has a direct application to multi-period portfolio selection, and this extension from a deterministic situation to a stochastic situation would be significant in terms of its academic value and its relevance to real applications. There is much room for the improvement of the branch and bound rules in the proposed algorithm. A tighter bound would speed up the fathoming process, which would thus relax the difficulty that arises from the combinatorial nature of the problem. An optimality condition for (Q_s) would greatly assist the design of a solution methodology. As can be observed in the numerical implementation, the algorithm often reaches the optimal solution very quickly as a result of the good initial solution. However, much more time is needed to check the optimality of these solutions and thus derivation of either the necessary or sufficient optimality condition for (Q_s) would be very helpful in the development of more efficient solution methods.

Bibliography

[1] B.D.O. Anderson and J.B. Moore, *Optimal Control: Linear Quadatric Methods*, Prentice Hall, Englewood Cliffs, NJ. 1990.

[2] Dimitri P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models*, Prentice Hall. 1976.

[3] D. Li, On the minimax solution of multiple linear-quadratic problems, *IEEE Trans, Automatic Control*, Vol. 35, 1990, pp. 1153-1156.

[4] D. Li, On general multiple linear-quadratic control problems, *IEEE Trans. Automatic Control*, Vol. 38, 1993, pp. 1722-1727.

[5] M.R. Garey and D.S. Johnson. *Computer and Intractibility, a Guide to the Theory of NP-Completeness*, W. H. Freeman and Co. San Francisco, 1979.

[6] L.A. Wolsey, *Integer Programming*, John Wiley and Sons Inc. 1998.

[7] X.P. Xu, Optimal control of switched systems: new results and open problems, *Proceeding of the American Control Conference 2000*, pp. 2683-2687.

[8] X.P. Xu, Optimal Control of Switched Systems Based on Parameterization of the Switching Instants, *IEEE Trans. Automatic Control*, Vol. 49, 2004, pp. 2-16.

[9] C.G. Cassandras et, A Hierarchical Decomposition Method for Optimal Control of Hybrid Systems, *Proceeding of the 39th IEEE Conference on Decision Control*, 2000, pp. 2472-2477.

[10] D. Bienstock, A computational study of a family of mixed-integer quadratic programming problems, *Mathematical Programming*, Vol. 74, 1996, pp. 121-140.

[11] T.J. Chang, N. Meade, J.E. Beasley, Y.M. Sharaiha, Heuristics for cardinality constrained portfolio optimisation, *Computers and Operations Research*, Vol. 27, 2000, pp. 1271-1302.

[12] Miller, A, Subset Selection in Regression, *Monographs on Statistics and Applied Probability 40*, Chapman and Hall, 1990.